
Subject: [PATCH 8/8] RSS accounting hooks over the code
Posted by [Pavel Emelianov](#) on Mon, 04 Jun 2007 13:45:40 GMT
[View Forum Message](#) <> [Reply to Message](#)

As described, pages are charged to their first touchers.
The first toucher is determined using pages' _mapcount manipulations in rmap calls.

A page is charged in two stages:

1. preparation, in which the resource availability is checked.
This stage may lead to page reclamation, thus it is performed in a "might-sleep" places;
2. the container assignment to page. This is done in an atomic code that handles races between multiple touchers.

The uncharge process is tricky. After the _mapcount is changed and happens to hit zero we cannot trust the page->container pointer any longer. So we get the pointer before the decrement.

Includes fixes from Balbir Singh <balbir@in.ibm.com>

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
diff -upr linux-2.6.22-rc2-mm1.orig/fs/exec.c linux-2.6.22-rc2-mm1-2/fs/exec.c
--- linux-2.6.22-rc2-mm1.orig/fs/exec.c 2007-05-30 12:32:34.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/fs/exec.c 2007-06-01 16:35:13.000000000 +0400
@@ -59,6 +59,8 @@
#include <linux/kmod.h>
#endif

+#include <linux/rss_container.h>
+
int core_uses_pid;
char core_pattern[CORENAME_MAX_SIZE] = "core";
int suid_dumpable = 0;
@@ -314,27 +316,34 @@ void install_arg_page(struct vm_area_str
struct mm_struct *mm = vma->vm_mm;
pte_t *pte;
spinlock_t *ptl;
+ struct page_container *pcont;

if (unlikely(anon_vma_prepare(vma)))
goto out;

+ if (container_rss_prepare(page, vma, &pcont))
+ goto out;
```

```

+
flush_dcache_page(page);
pte = get_locked_pte(mm, address, &ptl);
if (!pte)
- goto out;
+ goto out_release;
if (!pte_none(*pte)) {
    pte_unmap_unlock(pte, ptl);
- goto out;
+ goto out_release;
}
inc_mm_counter(mm, anon_rss);
lru_cache_add_active(page);
set_pte_at(mm, address, pte, pte_mkdirty(pte_mkwrite(mk_pte(
    page, vma->vm_page_prot))));
- page_add_new_anon_rmap(page, vma, address);
+ page_add_new_anon_rmap(page, vma, address, pcont);
pte_unmap_unlock(pte, ptl);

/* no need for flush_tlb */
return;
+
+out_release:
+ container_rss_release(pcont);
out:
__free_page(page);
force_sig(SIGKILL, current);
diff -upr linux-2.6.22-rc2-mm1.orig/include/linux/rmap.h
linux-2.6.22-rc2-mm1-2/include/linux/rmap.h
--- linux-2.6.22-rc2-mm1.orig/include/linux/rmap.h 2007-05-29 18:01:57.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/include/linux/rmap.h 2007-06-01 16:35:13.000000000 +0400
@@ -69,9 +69,13 @@ void __anon_vma_link(struct vm_area_stru
/*
 * rmap interfaces called when adding or removing pte of page
 */
-void page_add_anon_rmap(struct page *, struct vm_area_struct *, unsigned long);
-void page_add_new_anon_rmap(struct page *, struct vm_area_struct *, unsigned long);
-void page_add_file_rmap(struct page *);
+struct page_container;
+
+void page_add_anon_rmap(struct page *, struct vm_area_struct *,
+ unsigned long, struct page_container *);
+void page_add_new_anon_rmap(struct page *, struct vm_area_struct *,
+ unsigned long, struct page_container *);
+void page_add_file_rmap(struct page *, struct page_container *);
void page_remove_rmap(struct page *, struct vm_area_struct *);

#ifdef CONFIG_DEBUG_VM

```

```

diff -upr linux-2.6.22-rc2-mm1.orig/mm/memory.c linux-2.6.22-rc2-mm1-2/mm/memory.c
--- linux-2.6.22-rc2-mm1.orig/mm/memory.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/memory.c 2007-06-01 16:35:13.000000000 +0400
@@ -60,6 +60,8 @@
#include <linux/swapops.h>
#include <linux/elf.h>

+#include <linux/rss_container.h>
+
#ifdef CONFIG_NEED_MULTIPLE_NODES
/* use the per-pgdat data instead for discontigmem - mbligh */
unsigned long max_mapnr;
@@ -1155,7 +1157,7 @@ static int zeromap_pte_range(struct mm_s
    break;
}
    page_cache_get(page);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, NULL);
    inc_mm_counter(mm, file_rss);
    set_pte_at(mm, addr, pte, zero_pte);
} while (pte++, addr += PAGE_SIZE, addr != end);
@@ -1263,7 +1265,7 @@ static int insert_page(struct mm_struct
/* Ok, finally just insert the thing.. */
get_page(page);
inc_mm_counter(mm, file_rss);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, NULL);
set_pte_at(mm, addr, pte, mk_pte(page, prot));

    retval = 0;
@@ -1668,6 +1670,7 @@ static int do_wp_page(struct mm_struct *
    pte_t entry;
    int reuse = 0, ret = VM_FAULT_MINOR;
    struct page *dirty_page = NULL;
+ struct page_container *pcont;

    old_page = vm_normal_page(vma, address, orig_pte);
    if (!old_page)
@@ -1752,6 +1755,9 @@ gotten:
    cow_user_page(new_page, old_page, address, vma);
}

+ if (container_rss_prepare(new_page, vma, &pcont))
+ goto oom;
+
/*
 * Re-check the pte - we dropped the lock
 */

```

```

@@ -1779,12 +1785,14 @@ gotten:
    set_pte_at(mm, address, page_table, entry);
    update_mmu_cache(vma, address, entry);
    lru_cache_add_active(new_page);
- page_add_new_anon_rmap(new_page, vma, address);
+ page_add_new_anon_rmap(new_page, vma, address, pcont);

    /* Free the old page.. */
    new_page = old_page;
    ret |= VM_FAULT_WRITE;
- }
+ } else
+ container_rss_release(pcont);
+
    if (new_page)
        page_cache_release(new_page);
    if (old_page)
@@ -2176,6 +2184,7 @@ static int do_swap_page(struct mm_struct
    swp_entry_t entry;
    pte_t pte;
    int ret = VM_FAULT_MINOR;
+ struct page_container *pcont;

    if (!pte_unmap_same(mm, pmd, page_table, orig_pte))
        goto out;
@@ -2208,6 +2217,11 @@ static int do_swap_page(struct mm_struct
    count_vm_event(PGMAJFAULT);
}

+ if (container_rss_prepare(page, vma, &pcont)) {
+ ret = VM_FAULT_OOM;
+ goto out;
+ }
+
    delayacct_clear_flag(DELAYACCT_PF_SWAPIN);
    mark_page_accessed(page);
    lock_page(page);
@@ -2221,6 +2235,7 @@ static int do_swap_page(struct mm_struct

    if (unlikely(!PageUptodate(page))) {
        ret = VM_FAULT_SIGBUS;
+ container_rss_release(pcont);
        goto out_nomap;
    }

@@ -2235,7 +2250,7 @@ static int do_swap_page(struct mm_struct

    flush_icache_page(vma, page);

```

```

    set_pte_at(mm, address, page_table, pte);
- page_add_anon_rmap(page, vma, address);
+ page_add_anon_rmap(page, vma, address, pcont);

    swap_free(entry);
    if (vm_swap_full())
@@ -2256,6 +2271,7 @@ unlock:
out:
    return ret;
out_nomap:
+ container_rss_release(pcont);
    pte_unmap_unlock(page_table, ptl);
    unlock_page(page);
    page_cache_release(page);
@@ -2274,6 +2290,7 @@ static int do_anonymous_page(struct mm_s
    struct page *page;
    spinlock_t *ptl;
    pte_t entry;
+ struct page_container *pcont;

    if (write_access) {
        /* Allocate our own private page. */
@@ -2285,15 +2302,19 @@ static int do_anonymous_page(struct mm_s
        if (!page)
            goto oom;

+ if (container_rss_prepare(page, vma, &pcont))
+ goto oom_release;
+
        entry = mk_pte(page, vma->vm_page_prot);
        entry = maybe_mkdirty(entry, vma);

        page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
        if (!pte_none(*page_table))
- goto release;
+ goto release_container;
+
        inc_mm_counter(mm, anon_rss);
        lru_cache_add_active(page);
- page_add_new_anon_rmap(page, vma, address);
+ page_add_new_anon_rmap(page, vma, address, pcont);
    } else {
        /* Map the ZERO_PAGE - vm_page_prot is readonly */
        page = ZERO_PAGE(address);
@@ -2305,7 +2326,7 @@ static int do_anonymous_page(struct mm_s
        if (!pte_none(*page_table))
            goto release;
        inc_mm_counter(mm, file_rss);

```

```

- page_add_file_rmap(page);
+ page_add_file_rmap(page, NULL);
}

set_pte_at(mm, address, page_table, entry);
@@ -2316,9 +2337,14 @@ static int do_anonymous_page(struct mm_s
unlock:
pte_unmap_unlock(page_table, ptl);
return VM_FAULT_MINOR;
+release_container:
+ container_rss_release(pcont);
release:
page_cache_release(page);
goto unlock;
+
+oom_release:
+ page_cache_release(page);
oom:
return VM_FAULT_OOM;
}
@@ -2346,6 +2372,7 @@ static int __do_fault(struct mm_struct *
int anon = 0;
struct page *dirty_page = NULL;
struct fault_data fdata;
+ struct page_container *pcont;

fdata.address = address & PAGE_MASK;
fdata.pgoff = pgoff;
@@ -2415,6 +2442,11 @@ static int __do_fault(struct mm_struct *

}

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

/*
@@ -2437,10 +2469,10 @@ static int __do_fault(struct mm_struct *
if (anon) {
inc_mm_counter(mm, anon_rss);
lru_cache_add_active(page);
- page_add_new_anon_rmap(page, vma, address);
+ page_add_new_anon_rmap(page, vma, address, pcont);
} else {
inc_mm_counter(mm, file_rss);

```

```

- page_add_file_rmap(page);
+ page_add_file_rmap(page, pcont);
  if (flags & FAULT_FLAG_WRITE) {
    dirty_page = page;
    get_page(dirty_page);
@@ -2451,6 +2483,7 @@ static int __do_fault(struct mm_struct *
  update_mmu_cache(vma, address, entry);
  lazy_mmu_prot_update(entry);
  } else {
+ container_rss_release(pcont);
  if (anon)
    page_cache_release(page);
  else
diff -upr linux-2.6.22-rc2-mm1.orig/mm/migrate.c linux-2.6.22-rc2-mm1-2/mm/migrate.c
--- linux-2.6.22-rc2-mm1.orig/mm/migrate.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/migrate.c 2007-06-01 16:35:13.000000000 +0400
@@ -28,6 +28,7 @@
#include <linux/mempolicy.h>
#include <linux/vmalloc.h>
#include <linux/security.h>
+#include <linux/rss_container.h>

#include "internal.h"

@@ -134,6 +135,7 @@ static void remove_migration_pte(struct
  pte_t *ptep, pte;
  spinlock_t *ptl;
  unsigned long addr = page_address_in_vma(new, vma);
+ struct page_container *pcont;

  if (addr == -EFAULT)
    return;
@@ -157,6 +159,11 @@ static void remove_migration_pte(struct
  return;
}

+ if (container_rss_prepare(new, vma, &pcont)) {
+   pte_unmap(ptep);
+   return;
+ }
+
  ptl = pte_lockptr(mm, pmd);
  spin_lock(ptl);
  pte = *ptep;
@@ -175,6 +182,19 @@ static void remove_migration_pte(struct
  set_pte_at(mm, addr, ptep, pte);

  if (PageAnon(new))

```

```

- page_add_anon_rmap(new, vma, addr);
+ page_add_anon_rmap(new, vma, addr, pcont);
  else
- page_add_file_rmap(new);
+ page_add_file_rmap(new, pcont);

  /* No need to invalidate - it was non-present before */
  update_mmu_cache(vma, addr, pte);
  lazy_mmu_prot_update(pte);
+ pte_unmap_unlock(pte, pte);
+ return;

out:
  pte_unmap_unlock(pte, pte);
+ container_rss_release(pcont);
}

/*
diff -upr linux-2.6.22-rc2-mm1.orig/mm/page_alloc.c linux-2.6.22-rc2-mm1-2/mm/page_alloc.c
--- linux-2.6.22-rc2-mm1.orig/mm/page_alloc.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/page_alloc.c 2007-06-01 16:35:15.000000000 +0400
@@ -41,6 +41,7 @@
#include <linux/pfn.h>
#include <linux/backing-dev.h>
#include <linux/fault-inject.h>
+#include <linux/rss_container.h>

#include <asm/tlbflush.h>
#include <asm/div64.h>
@@ -1042,6 +1043,7 @@ static void fastcall free_hot_cold_page(

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

@@ -2592,6 +2594,7 @@ void __meminit memmap_init_zone(unsigned
  set_page_links(page, zone, nid, pfn);
  init_page_count(page);
  reset_page_mapcount(page);
+ set_page_container(page, NULL);
  SetPageReserved(page);

/*
diff -upr linux-2.6.22-rc2-mm1.orig/mm/rmap.c linux-2.6.22-rc2-mm1-2/mm/rmap.c
--- linux-2.6.22-rc2-mm1.orig/mm/rmap.c 2007-05-30 12:32:36.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/rmap.c 2007-06-01 16:35:15.000000000 +0400

```

@@ -51,6 +51,8 @@

```
#include <asm/tlbflush.h>
```

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

```
+
```

```
struct kmem_cache *anon_vma_cache;
```

```
static inline void validate_anon_vma(struct vm_area_struct *find_vma)
```

@@ -563,18 +565,28 @@ static void __page_check_anon_rmap(struct

```
* @page: the page to add the mapping to
```

```
* @vma: the vm area in which the mapping is added
```

```
* @address: the user virtual address mapped
```

```
+ * @pcont: the page beancounter to charge page with
```

```
*
```

```
* The caller needs to hold the pte lock and the page must be locked.
```

```
*/
```

```
void page_add_anon_rmap(struct page *page,
```

```
- struct vm_area_struct *vma, unsigned long address)
```

```
+ struct vm_area_struct *vma, unsigned long address,
```

```
+ struct page_container *pcont)
```

```
{
```

```
VM_BUG_ON(!PageLocked(page));
```

```
VM_BUG_ON(address < vma->vm_start || address >= vma->vm_end);
```

```
- if (atomic_inc_and_test(&page->_mapcount))
```

```
+ if (atomic_inc_and_test(&page->_mapcount)) {
```

```
__page_set_anon_rmap(page, vma, address);
```

```
- else
```

```
+ /* 0 -> 1 state change */
```

```
+ container_rss_add(pcont);
```

```
+ } else {
```

```
__page_check_anon_rmap(page, vma, address);
```

```
+ /*
```

```
+ * we raced with another touch or just mapped the page
```

```
+ * for the N-th time
```

```
+ */
```

```
+ container_rss_release(pcont);
```

```
+ }
```

```
}
```

```
/*
```

@@ -582,29 +594,44 @@ void page_add_anon_rmap(struct page *pag

```
* @page: the page to add the mapping to
```

```
* @vma: the vm area in which the mapping is added
```

```
* @address: the user virtual address mapped
```

```
+ * @pcont: the page beancounter to charge page with
```

```
*
```

```
* Same as page_add_anon_rmap but must only be called on *new* pages.
```

```

* This means the inc-and-test can be bypassed.
* Page does not have to be locked.
*/
void page_add_new_anon_rmap(struct page *page,
- struct vm_area_struct *vma, unsigned long address)
+ struct vm_area_struct *vma, unsigned long address,
+ struct page_container *pcont)
{
    BUG_ON(address < vma->vm_start || address >= vma->vm_end);
    atomic_set(&page->_mapcount, 0); /* elevate count by 1 (starts at -1) */
+ container_rss_add(pcont);
    __page_set_anon_rmap(page, vma, address);
}

/**
 * page_add_file_rmap - add pte mapping to a file page
- * @page: the page to add the mapping to
+ * @page: the page to add the mapping to
+ * @pcont: the page beancounter to charge page with
 *
 * The caller needs to hold the pte lock.
 */
-void page_add_file_rmap(struct page *page)
+void page_add_file_rmap(struct page *page, struct page_container *pcont)
{
- if (atomic_inc_and_test(&page->_mapcount))
+ if (atomic_inc_and_test(&page->_mapcount)) {
+ if (pcont)
+ /*
+  * 0 -> 1 state change. add the page
+  *
+  * some pages like ZERO_PAGE() or driver specific pages
+  * are not added to the container as they do not imply
+  * RSS consumption --xemul
+  */
+ container_rss_add(pcont);
    __inc_zone_page_state(page, NR_FILE_MAPPED);
+ } else if (pcont)
+ container_rss_release(pcont);
}

#ifdef CONFIG_DEBUG_VM
@@ -635,6 +662,15 @@ void page_dup_rmap(struct page *page, st
*/
void page_remove_rmap(struct page *page, struct vm_area_struct *vma)
{
+ struct page_container *pcont;
+

```

```

+ /*
+  * this place is tricky. we get the container pointer *before* the
+  * mapcount change. this is done so sice we cannot trust this pointer
+  * *after* mapcount drops to zero - the page can be re-mapped right
+  * after this so we risk getting the wrong container --xemul
+  */
+ pcont = page_container(page);
+ if (atomic_add_negative(-1, &page->_mapcount)) {
+   if (unlikely(page_mapcount(page) < 0)) {
+     printk (KERN_EMERG "Eeek! page_mapcount(page) went negative! (%d)\n",
page_mapcount(page));
@@ -652,6 +688,8 @@ void page_remove_rmap(struct page *page,
    BUG();
  }

+ if (pcont)
+   container_rss_del(pcont);
+ /*
+  * It would be tidy to reset the PageAnon mapping here,
+  * but that might overwrite a racing page_add_anon_rmap
diff -upr linux-2.6.22-rc2-mm1.orig/mm/swapfile.c linux-2.6.22-rc2-mm1-2/mm/swapfile.c
--- linux-2.6.22-rc2-mm1.orig/mm/swapfile.c 2007-05-11 16:36:58.000000000 +0400
+++ linux-2.6.22-rc2-mm1-2/mm/swapfile.c 2007-06-01 16:35:13.000000000 +0400
@@ -32,6 +32,8 @@
#include <asm/tlbflush.h>
#include <linux/swapops.h>

+#include <linux/rss_container.h>
+
DEFINE_SPINLOCK(swap_lock);
unsigned int nr_swapfiles;
long total_swap_pages;
@@ -507,13 +509,14 @@ unsigned int count_swap_pages(int type,
 * force COW, vm_page_prot omits write permission from any private vma.
 */
static void unuse_pte(struct vm_area_struct *vma, pte_t *pte,
- unsigned long addr, swp_entry_t entry, struct page *page)
+ unsigned long addr, swp_entry_t entry, struct page *page,
+ struct page_container *pcont)
{
  inc_mm_counter(vma->vm_mm, anon_rss);
  get_page(page);
  set_pte_at(vma->vm_mm, addr, pte,
    pte_mkold(mk_pte(page, vma->vm_page_prot)));
- page_add_anon_rmap(page, vma, addr);
+ page_add_anon_rmap(page, vma, addr, pcont);
  swap_free(entry);
+ /*

```

```

    * Move the page to the active list so it is not
@@ -530,6 +533,10 @@ static int unuse_pte_range(struct vm_are
    pte_t *pte;
    spinlock_t *ptl;
    int found = 0;
+ struct page_container *pcont;
+
+ if (container_rss_prepare(page, vma, &pcont))
+ return 0;

    pte = pte_offset_map_lock(vma->vm_mm, pmd, addr, &ptl);
    do {
@@ -538,12 +545,14 @@ 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);
+ unuse_pte(vma, pte++, addr, entry, page, pcont);
        found = 1;
        break;
    }
    } while (pte++, addr += PAGE_SIZE, addr != end);
    pte_unmap_unlock(pte - 1, ptl);
+ if (!found)
+ container_rss_release(pcont);
    return found;
}

```
