Subject: [RFC][PATCH 4/7] RSS accounting hooks over the code
Posted by xemul on Tue, 06 Mar 2007 14:57:51 GMT
View Forum Message <> Reply to Message

Pages are charged to their first touchers which are
determined using pages' mapcount manipulations in
rmap calls.

```
diff -upr linux-2.6.20.orig/fs/exec.c linux-2.6.20-0/fs/exec.c
--- linux-2.6.20.orig/fs/exec.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/fs/exec.c 2007-03-06 13:33:28.000000000 +0300
@@ -58,6 +58,8 @@
 #include <linux/kmod.h>
 #endif

+#include <linux/rss_container.h>
+
 int core_uses_pid;
 char core_pattern[128] = "core";
 int suid_dumpable = 0;
@@ -309,27 +311,34 @@ 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-2.6.20.orig/include/linux/rmap.h linux-2.6.20-0/include/linux/rmap.h
--- linux-2.6.20.orig/include/linux/rmap.h 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/include/linux/rmap.h 2007-03-06 13:33:28.000000000 +0300
@@ -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 *);

 /**
diff -upr linux-2.6.20.orig/mm/fremap.c linux-2.6.20-0/mm/fremap.c
--- linux-2.6.20.orig/mm/fremap.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/fremap.c 2007-03-06 13:33:28.000000000 +0300
@@ -20,6 +20,8 @@
 #include <asm/cacheflush.h>
 #include <asm/tlbflush.h>

+#include <linux/rss_container.h>
+
 static int zap_pte(struct mm_struct *mm, struct vm_area_struct *vma,
    unsigned long addr, pte_t *ptep)
 {
@@ -57,6 +59,10 @@ int install_page(struct mm_struct *mm, s
  pte_t *pte;
  pte_t pte_val;
  spinlock_t *ptl;
+ struct page_container *pcont;
+
```

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

  pte = get_locked_pte(mm, addr, &ptl);
  if (!pte)
@@ -81,13 +87,16 @@ int install_page(struct mm_struct *mm, s
  flush_icache_page(vma, page);
  pte_val = mk_pte(page, prot);
  set_pte_at(mm, addr, pte, pte_val);
- page_add_file_rmap(page);
+ page_add_file_rmap(page, pcont);
  update_mmu_cache(vma, addr, pte_val);
  lazy_mmu_prot_update(pte_val);
  err = 0;
 unlock:
  pte_unmap_unlock(pte, ptl);
 out:
+ if (err != 0)
+  container_rss_release(pcont);
+out_release:
  return err;
 }
 EXPORT_SYMBOL(install_page);
diff -upr linux-2.6.20.orig/mm/memory.c linux-2.6.20-0/mm/memory.c
--- linux-2.6.20.orig/mm/memory.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/memory.c 2007-03-06 13:33:28.000000000 +0300
@@ -60,6 +60,8 @@
 #include <linux/swapops.h>
 #include <linux/elf.h>

+#include <linux/rss_container.h>
+
 #ifndef CONFIG_NEED_MULTIPLE_NODES
 /* use the per-pgdat data instead for discontigmem - mbligh */
 unsigned long max_mapnr;
@@ -1126,7 +1128,7 @@ 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);
@@ -1234,7 +1236,7 @@ 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;
@@ -1495,6 +1497,7 @@ 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)
@@ -1580,6 +1583,9 @@ 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
   */
@@ -1607,12 +1613,14 @@ 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)
@@ -1988,6 +1996,7 @@ 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;
```

```
@@ -2020,6 +2029,11 @@ 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);
@@ -2033,6 +2047,7 @@ static int do_swap_page(struct mm_struct

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

@@ -2047,7 +2062,7 @@ 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())
@@ -2069,6 +2084,7 @@ unlock:
 out:
  return ret;
 out_nomap:
+ container_rss_release(pcont);
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
  page_cache_release(page);
@@ -2087,6 +2103,7 @@ 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. */
@@ -2098,15 +2115,19 @@ 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_mkwrite(pte_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);
@@ -2118,7 +2139,7 @@ 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);
@@ -2129,9 +2150,14 @@ 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;
 }
@@ -2161,6 +2187,7 @@ static int do_no_page(struct mm_struct *
  int ret = VM_FAULT_MINOR;
  int anon = 0;
  struct page *dirty_page = NULL;
+ struct page_container *pcont;
```

```
  pte_unmap(page_table);
  BUG_ON(vma->vm_flags & VM_PFNMAP);
@@ -2218,6 +2245,9 @@ retry:
    }
   }

+ if (container_rss_prepare(new_page, vma, &pcont))
+  goto oom;
+
   page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
   /*
    * For a file-backed vma, someone could have truncated or otherwise
@@ -2226,6 +2256,7 @@ retry:
    */
   if (mapping && unlikely(sequence != mapping->truncate_count)) {
    pte_unmap_unlock(page_table, ptl);
+  container_rss_release(pcont);
    page_cache_release(new_page);
    cond_resched();
    sequence = mapping->truncate_count;
@@ -2253,10 +2284,10 @@ retry:
    if (anon) {
     inc_mm_counter(mm, anon_rss);
     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);
    } else {
     inc_mm_counter(mm, file_rss);
-   page_add_file_rmap(new_page);
+    page_add_file_rmap(new_page, pcont);
     if (write_access) {
      dirty_page = new_page;
      get_page(dirty_page);
@@ -2264,6 +2295,7 @@ retry:
    }
   } else {
    /* One of our sibling threads was faster, back out. */
+  container_rss_release(pcont);
    page_cache_release(new_page);
    goto unlock;
   }
diff -upr linux-2.6.20.orig/mm/migrate.c linux-2.6.20-0/mm/migrate.c
--- linux-2.6.20.orig/mm/migrate.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/migrate.c 2007-03-06 13:33:28.000000000 +0300
@@ -134,6 +134,7 @@ 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 +158,11 @@ 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;
@@ -175,16 +181,19 @@ 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(ptep, ptl);
+ return;

 out:
  pte_unmap_unlock(ptep, ptl);
+ container_rss_release(pcont);
 }

 /*
diff -upr linux-2.6.20.orig/mm/rmap.c linux-2.6.20-0/mm/rmap.c
--- linux-2.6.20.orig/mm/rmap.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/rmap.c 2007-03-06 13:33:28.000000000 +0300
@@ -51,6 +51,8 @@

 #include <asm/tlbflush.h>

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

```
 struct kmem_cache *anon_vma_cachep;

 static inline void validate_anon_vma(struct vm_area_struct *find_vma)
@@ -526,14 +528,19 @@ static void __page_set_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.
  */
 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)
 {
- if (atomic_inc_and_test(&page->_mapcount))
+ if (atomic_inc_and_test(&page->_mapcount)) {
+  container_rss_add(pcont);
   __page_set_anon_rmap(page, vma, address);
+ } else
+  container_rss_release(pcont);
  /* else checking page index and mapping is racy */
 }

@@ -542,27 +549,35 @@ 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.
  */
 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)
 {
  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)
+   container_rss_add(pcont);
    __inc_zone_page_state(page, NR_FILE_MAPPED);
+ } else if (pcont)
+  container_rss_release(pcont);
 }

 /**
@@ -573,6 +588,9 @@ void page_add_file_rmap(struct page *pag
  */
 void page_remove_rmap(struct page *page, struct vm_area_struct *vma)
 {
+ struct page_container *pcont;
+
+ 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));
@@ -588,6 +606,8 @@ 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-2.6.20.orig/mm/swapfile.c linux-2.6.20-0/mm/swapfile.c
--- linux-2.6.20.orig/mm/swapfile.c 2007-02-04 21:44:54.000000000 +0300
+++ linux-2.6.20-0/mm/swapfile.c 2007-03-06 13:33:28.000000000 +0300
@@ -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;
 }
```