
Subject: [PATCH 13/13] BC: vmrss (charges)
Posted by [dev](#) on Tue, 05 Sep 2006 15:29:59 GMT
[View Forum Message](#) <> [Reply to Message](#)

Introduce calls to BC code over the kernel to add
accounting of physical pages/privvmpages.

Signed-Off-By: Pavel Emelianov <xemul@sw.ru>
Signed-Off-By: Kirill Korotaev <dev@sw.ru>

```
fs/exec.c      | 11 ++++
include/linux/mm.h | 3 -
kernel/fork.c   | 2
mm/filemap_xip.c | 2
mm/fremap.c     | 11 ++++
mm/memory.c     | 141 ++++++++++++++++++++++++++++++++++++++-----
mm/migrate.c    | 3 +
mm/mprotect.c   | 12 +++-
mm/rmap.c       | 4 +
mm/swapfile.c   | 47 ++++++++-----
10 files changed, 186 insertions(+), 50 deletions(-)
```

```
--- ./fs/exec.c.bcrssch 2006-09-05 12:53:55.000000000 +0400
+++ ./fs/exec.c 2006-09-05 13:51:55.000000000 +0400
@@ -50,6 +50,8 @@
#include <linux/cn_proc.h>
#include <linux/audit.h>
```

```
+#include <bc/vmrss.h>
+
#include <asm/uaccess.h>
#include <asm/mmu_context.h>
```

```
@@ -308,6 +310,11 @@ void install_arg_page(struct vm_area_str
    struct mm_struct *mm = vma->vm_mm;
    pte_t *pte;
    spinlock_t *ptl;
+ struct page_beancounter *pb;
+
+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+ goto out_nopb;

    if (unlikely(anon_vma_prepare(vma)))
        goto out;
@@ -325,11 +332,15 @@ void install_arg_page(struct vm_area_str
```

```

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);
+ bc_vmrss_page_add(page, mm, vma, &pb);
pte_unmap_unlock(pte, ptl);

/* no need for flush_tlb */
+ bc_free_rss_counter(pb);
return;
out:
+ bc_free_rss_counter(pb);
+out_nopb:
__free_page(page);
force_sig(SIGKILL, current);
}
--- ./include/linux/mm.h.bcrssch 2006-09-05 13:47:12.000000000 +0400
+++ ./include/linux/mm.h 2006-09-05 13:51:55.000000000 +0400
@@ -753,7 +753,8 @@ void free_pgd_range(struct mmu_gather **
void free_pgtables(struct mmu_gather **tlb, struct vm_area_struct *start_vma,
    unsigned long floor, unsigned long ceiling);
int copy_page_range(struct mm_struct *dst, struct mm_struct *src,
- struct vm_area_struct *vma);
+ struct vm_area_struct *vma,
+ struct vm_area_struct *dst_vma);
int zeromap_page_range(struct vm_area_struct *vma, unsigned long from,
    unsigned long size, pgprot_t prot);
void unmap_mapping_range(struct address_space *mapping,
--- ./kernel/fork.c.bcrssch 2006-09-05 13:23:27.000000000 +0400
+++ ./kernel/fork.c 2006-09-05 13:51:55.000000000 +0400
@@ -280,7 +280,7 @@ static inline int dup_mmap(struct mm_str
    rb_parent = &tmp->vm_rb;

    mm->map_count++;
- retval = copy_page_range(mm, oldmm, mpnt);
+ retval = copy_page_range(mm, oldmm, mpnt, tmp);

    if (tmp->vm_ops && tmp->vm_ops->open)
        tmp->vm_ops->open(tmp);
--- ./mm/filemap_xip.c.bcrssch 2006-07-10 12:39:20.000000000 +0400
+++ ./mm/filemap_xip.c 2006-09-05 13:51:55.000000000 +0400
@@ -13,6 +13,7 @@
#include <linux/module.h>
#include <linux/uio.h>
#include <linux/rmap.h>
+ #include <bc/vmrss.h>
#include <asm/tlbflush.h>
#include "filemap.h"

```

```

@@ -189,6 +190,7 @@ __xip_unmap (struct address_space * mapp
/* Nuke the page table entry. */
flush_cache_page(vma, address, pte_pfn(*pte));
pteval = ptep_clear_flush(vma, address, pte);
+ bc_vmrss_page_del(page, mm, vma);
page_remove_rmap(page);
dec_mm_counter(mm, file_rss);
BUG_ON(pte_dirty(pteval));
--- ./mm/fremap.c.bcrssch 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/fremap.c 2006-09-05 13:51:55.000000000 +0400
@@ -16,6 +16,8 @@
#include <linux/module.h>
#include <linux/syscalls.h>

+#include <bc/vmrss.h>
+
#include <asm/mmu_context.h>
#include <asm/cacheflush.h>
#include <asm/tlbflush.h>
@@ -33,6 +35,7 @@ static int zap_pte(struct mm_struct *mm,
if (page) {
if (pte_dirty(pte))
set_page_dirty(page);
+ bc_vmrss_page_del(page, mm, vma);
page_remove_rmap(page);
page_cache_release(page);
}
@@ -57,6 +60,11 @@ int install_page(struct mm_struct *mm, s
pte_t *pte;
pte_t pte_val;
spinlock_t *ptl;
+ struct page_beancounter *pb;
+
+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+ goto out_nopb;

pte = get_locked_pte(mm, addr, &ptl);
if (!pte)
@@ -82,12 +90,15 @@ int install_page(struct mm_struct *mm, s
pte_val = mk_pte(page, prot);
set_pte_at(mm, addr, pte, pte_val);
page_add_file_rmap(page);
+ bc_vmrss_page_add(page, mm, vma, &pb);
update_mmu_cache(vma, addr, pte_val);
lazy_mmu_prot_update(pte_val);
err = 0;
unlock:

```

```

    pte_unmap_unlock(pte, ptl);
out:
+ bc_free_rss_counter(pb);
+out_nopb:
    return err;
}
EXPORT_SYMBOL(install_page);
--- ./mm/memory.c.bcrssch 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/memory.c 2006-09-05 13:51:55.000000000 +0400
@@ -51,6 +51,9 @@
#include <linux/init.h>
#include <linux/writeback.h>

+#include <bc/vmpages.h>
+#include <bc/vmrss.h>
+
#include <asm/pgalloc.h>
#include <asm/uaccess.h>
#include <asm/tlb.h>
@@ -427,7 +430,9 @@ struct page *vm_normal_page(struct vm_ar
static inline void
copy_one_pte(struct mm_struct *dst_mm, struct mm_struct *src_mm,
    pte_t *dst_pte, pte_t *src_pte, struct vm_area_struct *vma,
- unsigned long addr, int *rss)
+ unsigned long addr, int *rss,
+ struct vm_area_struct *dst_vma,
+ struct page_beancounter **ppb)
{
    unsigned long vm_flags = vma->vm_flags;
    pte_t pte = *src_pte;
@@ -481,6 +486,7 @@ copy_one_pte(struct mm_struct *dst_mm, s
    page = vm_normal_page(vma, addr, pte);
    if (page) {
        get_page(page);
+ bc_vmrss_page_dup(page, dst_mm, dst_vma, ppb);
        page_dup_rmap(page);
        rss[!!PageAnon(page)]++;
    }
@@ -489,20 +495,32 @@ out_set_pte:
    set_pte_at(dst_mm, addr, dst_pte, pte);
}

+#define pte_ptr(a)    (PTRS_PER_PTE - ((a >> PAGE_SHIFT)&(PTRS_PER_PTE - 1)))
+
static int copy_pte_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
    pmd_t *dst_pmd, pmd_t *src_pmd, struct vm_area_struct *vma,
- unsigned long addr, unsigned long end)
+ unsigned long addr, unsigned long end,

```

```

+ struct vm_area_struct *dst_vma)
{
    pte_t *src_pte, *dst_pte;
    spinlock_t *src_ptl, *dst_ptl;
    int progress = 0;
- int rss[2];
+ int rss[2], err;
+ struct page_becounter *pb;

+ err = -ENOMEM;
+ pb = (mm_same_bc(dst_mm, src_mm) ? PB_COPY_SAME : NULL);
again:
+ if (pb != PB_COPY_SAME) {
+ pb = bc_alloc_rss_counter_list(pte_ptrs(addr), pb);
+ if (IS_ERR(pb))
+ goto out;
+ }
+
    rss[1] = rss[0] = 0;
    dst_pte = pte_alloc_map_lock(dst_mm, dst_pmd, addr, &dst_ptl);
    if (!dst_pte)
- return -ENOMEM;
+ goto out;
    src_pte = pte_offset_map_nested(src_pmd, addr);
    src_ptl = pte_lockptr(src_mm, src_pmd);
    spin_lock_nested(src_ptl, SINGLE_DEPTH_NESTING);
@@ -524,7 +542,8 @@ again:
    progress++;
    continue;
}
- copy_one_pte(dst_mm, src_mm, dst_pte, src_pte, vma, addr, rss);
+ copy_one_pte(dst_mm, src_mm, dst_pte, src_pte, vma, addr, rss,
+ dst_vma, &pb);
    progress += 8;
} while (dst_pte++, src_pte++, addr += PAGE_SIZE, addr != end);

@@ -536,12 +555,18 @@ again:
    cond_resched();
    if (addr != end)
        goto again;
- return 0;
+
+ err = 0;
+out:
+ if (pb != PB_COPY_SAME)
+ bc_free_rss_counter(pb);
+ return err;
}

```

```

static inline int copy_pmd_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
    pud_t *dst_pud, pud_t *src_pud, struct vm_area_struct *vma,
- unsigned long addr, unsigned long end)
+ unsigned long addr, unsigned long end,
+ struct vm_area_struct *dst_vma)
{
    pmd_t *src_pmd, *dst_pmd;
    unsigned long next;
@@ -555,7 +580,7 @@ static inline int copy_pmd_range(struct
    if (pmd_none_or_clear_bad(src_pmd))
        continue;
    if (copy_pte_range(dst_mm, src_mm, dst_pmd, src_pmd,
-    vma, addr, next))
+    vma, addr, next, dst_vma))
        return -ENOMEM;
    } while (dst_pmd++, src_pmd++, addr = next, addr != end);
    return 0;
@@ -563,7 +588,8 @@ static inline int copy_pmd_range(struct

static inline int copy_pud_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
    pgd_t *dst_pgd, pgd_t *src_pgd, struct vm_area_struct *vma,
- unsigned long addr, unsigned long end)
+ unsigned long addr, unsigned long end,
+ struct vm_area_struct *dst_vma)
{
    pud_t *src_pud, *dst_pud;
    unsigned long next;
@@ -577,14 +603,14 @@ static inline int copy_pud_range(struct
    if (pud_none_or_clear_bad(src_pud))
        continue;
    if (copy_pmd_range(dst_mm, src_mm, dst_pud, src_pud,
-    vma, addr, next))
+    vma, addr, next, dst_vma))
        return -ENOMEM;
    } while (dst_pud++, src_pud++, addr = next, addr != end);
    return 0;
}

int copy_page_range(struct mm_struct *dst_mm, struct mm_struct *src_mm,
- struct vm_area_struct *vma)
+ struct vm_area_struct *vma, struct vm_area_struct *dst_vma)
{
    pgd_t *src_pgd, *dst_pgd;
    unsigned long next;
@@ -612,7 +638,7 @@ int copy_page_range(struct mm_struct *ds
    if (pgd_none_or_clear_bad(src_pgd))
        continue;

```

```

    if (copy_pud_range(dst_mm, src_mm, dst_pgd, src_pgd,
-       vma, addr, next))
+       vma, addr, next, dst_vma))
        return -ENOMEM;
    } while (dst_pgd++, src_pgd++, addr = next, addr != end);
    return 0;
@@ -681,6 +707,7 @@ static unsigned long zap_pte_range(struct
    mark_page_accessed(page);
    file_rss--;
}
+ bc_vmrss_page_del(page, mm, vma);
  page_remove_rmap(page);
  tlb_remove_page(tlb, page);
  continue;
@@ -1104,8 +1131,9 @@ int get_user_pages(struct task_struct *t
}
EXPORT_SYMBOL(get_user_pages);

-static int zeromap_pte_range(struct mm_struct *mm, pmd_t *pmd,
- unsigned long addr, unsigned long end, pgprot_t prot)
+static int zeromap_pte_range(struct mm_struct *mm,
+ struct vm_area_struct *vma, pmd_t *pmd,
+ unsigned long addr, unsigned long end, pgprot_t prot)
{
    pte_t *pte;
    spinlock_t *ptl;
@@ -1118,6 +1146,7 @@ static int zeromap_pte_range(struct mm_s
    struct page *page = ZERO_PAGE(addr);
    pte_t zero_pte = pte_wrprotect(mk_pte(page, prot));
    page_cache_get(page);
+ bc_vmrss_page_add_noref(page, mm, vma);
    page_add_file_rmap(page);
    inc_mm_counter(mm, file_rss);
    BUG_ON(!pte_none(*pte));
@@ -1128,8 +1157,9 @@ static int zeromap_pte_range(struct mm_s
    return 0;
}

-static inline int zeromap_pmd_range(struct mm_struct *mm, pud_t *pud,
- unsigned long addr, unsigned long end, pgprot_t prot)
+static inline int zeromap_pmd_range(struct mm_struct *mm,
+ struct vm_area_struct *vma, pud_t *pud,
+ unsigned long addr, unsigned long end, pgprot_t prot)
{
    pmd_t *pmd;
    unsigned long next;
@@ -1139,14 +1169,15 @@ static inline int zeromap_pmd_range(stru
    return -ENOMEM;
}

```

```

do {
    next = pmd_addr_end(addr, end);
-   if (zeromap_pte_range(mm, pmd, addr, next, prot))
+   if (zeromap_pte_range(mm, vma, pmd, addr, next, prot))
        return -ENOMEM;
    } while (pmd++, addr = next, addr != end);
    return 0;
}

-static inline int zeromap_pud_range(struct mm_struct *mm, pgd_t *pgd,
-   unsigned long addr, unsigned long end, pgprot_t prot)
+static inline int zeromap_pud_range(struct mm_struct *mm,
+   struct vm_area_struct *vma, pgd_t *pgd,
+   unsigned long addr, unsigned long end, pgprot_t prot)
{
    pud_t *pud;
    unsigned long next;
@@ -1156,7 +1187,7 @@ static inline int zeromap_pud_range(stru
    return -ENOMEM;
    do {
        next = pud_addr_end(addr, end);
-   if (zeromap_pmd_range(mm, pud, addr, next, prot))
+   if (zeromap_pmd_range(mm, vma, pud, addr, next, prot))
        return -ENOMEM;
    } while (pud++, addr = next, addr != end);
    return 0;
@@ -1176,7 +1207,7 @@ int zeromap_page_range(struct vm_area_st
    flush_cache_range(vma, addr, end);
    do {
        next = pgd_addr_end(addr, end);
-   err = zeromap_pud_range(mm, pgd, addr, next, prot);
+   err = zeromap_pud_range(mm, vma, pgd, addr, next, prot);
        if (err)
            break;
    } while (pgd++, addr = next, addr != end);
@@ -1202,12 +1233,15 @@ pte_t *fastcall get_locked_pte(struct m
    * old drivers should use this, and they needed to mark their
    * pages reserved for the old functions anyway.
    */
-static int insert_page(struct mm_struct *mm, unsigned long addr, struct page *page, pgprot_t
prot)
+static int insert_page(struct vm_area_struct *vma, unsigned long addr, struct page *page,
pgprot_t prot)
{
+   struct mm_struct *mm;
    int retval;
    pte_t *pte;
    spinlock_t *ptl;

```

```

+ mm = vma->vm_mm;
+
  retval = -EINVAL;
  if (PageAnon(page))
    goto out;
@@ -1223,6 +1257,7 @@ static int insert_page(struct mm_struct
/* Ok, finally just insert the thing.. */
  get_page(page);
  inc_mm_counter(mm, file_rss);
+ bc_vmrss_page_add_noref(page, mm, vma);
  page_add_file_rmap(page);
  set_pte_at(mm, addr, pte, mk_pte(page, prot));

@@ -1262,7 +1297,7 @@ int vm_insert_page(struct vm_area_struct
  if (!page_count(page))
    return -EINVAL;
  vma->vm_flags |= VM_INSERTPAGE;
- return insert_page(vma->vm_mm, addr, page, vma->vm_page_prot);
+ return insert_page(vma, addr, page, vma->vm_page_prot);
}
EXPORT_SYMBOL(vm_insert_page);

@@ -1483,6 +1518,7 @@ 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_beancounter *pb;

  old_page = vm_normal_page(vma, address, orig_pte);
  if (!old_page)
@@ -1555,6 +1591,10 @@ static int do_wp_page(struct mm_struct *
gotten:
  pte_unmap_unlock(page_table, ptl);

+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+   goto oom_nopb;
+
  if (unlikely(anon_vma_prepare(vma)))
    goto oom;
  if (old_page == ZERO_PAGE(address)) {
@@ -1574,6 +1614,7 @@ gotten:
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
  if (likely(pte_same(*page_table, orig_pte))) {
    if (old_page) {
+   bc_vmrss_page_del(old_page, mm, vma);
    page_remove_rmap(old_page);

```

```

    if (!PageAnon(old_page)) {
        dec_mm_counter(mm, file_rss);
@@ -1589,6 +1630,7 @@ gotten:
        update_mmu_cache(vma, address, entry);
        lru_cache_add_active(new_page);
        page_add_new_anon_rmap(new_page, vma, address);
+ bc_vmrss_page_add(new_page, mm, vma, &pb);

    /* Free the old page.. */
    new_page = old_page;
@@ -1598,6 +1640,7 @@ gotten:
    page_cache_release(new_page);
    if (old_page)
        page_cache_release(old_page);
+ bc_free_rss_counter(pb);
unlock:
    pte_unmap_unlock(page_table, ptl);
    if (dirty_page) {
@@ -1606,6 +1649,8 @@ unlock:
    }
    return ret;
oom:
+ bc_free_rss_counter(pb);
+oom_nopb:
    if (old_page)
        page_cache_release(old_page);
    return VM_FAULT_OOM;
@@ -1970,9 +2015,14 @@ static int do_swap_page(struct mm_struct
    swp_entry_t entry;
    pte_t pte;
    int ret = VM_FAULT_MINOR;
+ struct page_beancounter *pb;

    if (!pte_unmap_same(mm, pmd, page_table, orig_pte))
- goto out;
+ goto out_nopb;
+
+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+ goto out_nopb;

    entry = pte_to_swp_entry(orig_pte);
    if (is_migration_entry(entry)) {
@@ -2030,6 +2080,7 @@ 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);
+ bc_vmrss_page_add(page, mm, vma, &pb);

```

```

    swap_free(entry);
    if (vm_swap_full())
@@ -2049,11 +2100,14 @@ static int do_swap_page(struct mm_struct
unlock:
    pte_unmap_unlock(page_table, ptl);
out:
+ bc_free_rss_counter(pb);
+out_nopb:
    return ret;
out_nomap:
    pte_unmap_unlock(page_table, ptl);
    unlock_page(page);
    page_cache_release(page);
+ bc_free_rss_counter(pb);
    return ret;
}

@@ -2069,11 +2123,16 @@ static int do_anonymous_page(struct mm_s
    struct page *page;
    spinlock_t *ptl;
    pte_t entry;
+ struct page_beancounter *pb;

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

+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+     goto oom_nopb;
+
        if (unlikely(anon_vma_prepare(vma)))
            goto oom;
        page = alloc_zeroed_user_highpage(vma, address);
@@ -2089,7 +2148,9 @@ static int do_anonymous_page(struct mm_s
        inc_mm_counter(mm, anon_rss);
        lru_cache_add_active(page);
        page_add_new_anon_rmap(page, vma, address);
+ bc_vmrss_page_add(page, mm, vma, &pb);
    } else {
+ pb = NULL;
        /* Map the ZERO_PAGE - vm_page_prot is readonly */
        page = ZERO_PAGE(address);
        page_cache_get(page);
@@ -2101,6 +2162,7 @@ static int do_anonymous_page(struct mm_s
        goto release;
        inc_mm_counter(mm, file_rss);

```

```

    page_add_file_rmap(page);
+ bc_vmrss_page_add_noref(page, mm, vma);
}

    set_pte_at(mm, address, page_table, entry);
@@ -2110,11 +2172,14 @@ static int do_anonymous_page(struct mm_s
    lazy_mmu_prot_update(entry);
unlock:
    pte_unmap_unlock(page_table, ptl);
+ bc_free_rss_counter(pb);
    return VM_FAULT_MINOR;
release:
    page_cache_release(page);
    goto unlock;
oom:
+ bc_free_rss_counter(pb);
+oom_nopb:
    return VM_FAULT_OOM;
}

@@ -2143,6 +2208,7 @@ static int do_no_page(struct mm_struct *
    int ret = VM_FAULT_MINOR;
    int anon = 0;
    struct page *dirty_page = NULL;
+ struct page_beancounter *pb;

    pte_unmap(page_table);
    BUG_ON(vma->vm_flags & VM_PFNMAP);
@@ -2152,6 +2218,10 @@ static int do_no_page(struct mm_struct *
    sequence = mapping->truncate_count;
    smp_rmb(); /* serializes i_size against truncate_count */
}
+
+ pb = bc_alloc_rss_counter();
+ if (IS_ERR(pb))
+ goto oom_nopb;
retry:
    new_page = vma->vm_ops->nopage(vma, address & PAGE_MASK, &ret);
    /*
@@ -2164,9 +2234,9 @@ retry:

    /* no page was available -- either SIGBUS or OOM */
    if (new_page == NOPAGE_SIGBUS)
- return VM_FAULT_SIGBUS;
+ goto bus_nopg;
    if (new_page == NOPAGE_OOM)
- return VM_FAULT_OOM;
+ goto oom_nopg;

```

```

/*
 * Should we do an early C-O-W break?
@@ -2190,11 +2260,8 @@ retry:
 * address space wants to know that the page is about
 * to become writable */
if (vma->vm_ops->page_mkwrite &&
-   vma->vm_ops->page_mkwrite(vma, new_page) < 0
-   ) {
-   page_cache_release(new_page);
-   return VM_FAULT_SIGBUS;
-   }
+   vma->vm_ops->page_mkwrite(vma, new_page) < 0)
+   goto bus;
+   }
+   }

@@ -2242,6 +2309,8 @@ retry:
    get_page(dirty_page);
    }
    }
+
+   bc_vmrss_page_add(new_page, mm, vma, &pb);
+   } else {
    /* One of our sibling threads was faster, back out. */
    page_cache_release(new_page);
@@ -2257,10 +2326,20 @@ unlock:
    set_page_dirty_balance(dirty_page);
    put_page(dirty_page);
    }
+   bc_free_rss_counter(pb);
+   return ret;
+   oom:
+   page_cache_release(new_page);
+oom_nopg:
+   bc_free_rss_counter(pb);
+oom_nopb:
+   return VM_FAULT_OOM;
+
+bus:
+   page_cache_release(new_page);
+bus_nopg:
+   bc_free_rss_counter(pb);
+   return VM_FAULT_SIGBUS;
+   }

/*
--- ./mm/migrate.c.bcrssch 2006-09-05 12:53:59.000000000 +0400

```

```

+++ ./mm/migrate.c 2006-09-05 13:51:55.000000000 +0400
@@ -29,6 +29,8 @@
#include <linux/vmalloc.h>
#include <linux/security.h>

+#include <bc/vmrss.h>
+
#include "internal.h"

#define lru_to_page(_head) (list_entry((_head)->prev, struct page, lru))
@@ -179,6 +181,7 @@ static void remove_migration_pte(struct
else
page_add_file_rmap(new);

+ bc_vmrss_page_del(new, mm, vma);
/* No need to invalidate - it was non-present before */
update_mmu_cache(vma, addr, pte);
lazy_mmu_prot_update(pte);
--- ./mm/mprotect.c.bcrssch 2006-09-05 13:27:40.000000000 +0400
+++ ./mm/mprotect.c 2006-09-05 13:54:20.000000000 +0400
@@ -22,6 +22,7 @@
#include <linux/swap.h>
#include <linux/swapops.h>
#include <bc/vmpages.h>
+#include <bc/vmrss.h>
#include <asm/uaccess.h>
#include <asm/pgtable.h>
#include <asm/cacheflush.h>
@@ -141,6 +142,7 @@ mprotect_fixup(struct vm_area_struct *vm
int error;
int dirty_accountable = 0;
int recharge;
+ unsigned long rss;

if (newflags == oldflags) {
*pprev = vma;
@@ -148,8 +150,10 @@ mprotect_fixup(struct vm_area_struct *vm
}

recharge = bc_privvm_recharge(oldflags, newflags, vma->vm_file);
- if (recharge == BC_CHARGE) {
- if (bc_privvm_charge(mm, end - start))
+ if (recharge != BC_NOCHARGE) {
+ rss = mm_rss_pages(mm, start, end) << PAGE_SHIFT;
+ if (recharge == BC_CHARGE && bc_privvm_charge(mm,
+ end - start - rss) < 0)
return -ENOMEM;
}
}

```

```
@@ -215,7 +219,7 @@ success:
    change_protection(vma, start, end, vma->vm_page_prot, dirty_accountable);
```

```
    if (recharge == BC_UNCHARGE)
-   bc_privvm_uncharge(mm, end - start);
+   bc_privvm_uncharge(mm, end - start - rss);
    vm_stat_account(mm, oldflags, vma->vm_file, -nrpages);
    vm_stat_account(mm, newflags, vma->vm_file, nrpages);
    return 0;
```

```
@@ -224,7 +228,7 @@ fail:
    vm_unacct_memory(charged);
fail_acct:
    if (recharge == BC_CHARGE)
-   bc_privvm_uncharge(mm, end - start);
+   bc_privvm_uncharge(mm, end - start - rss);
    return error;
}
```

```
--- ./mm/rmap.c.bcrssch 2006-09-05 12:58:17.000000000 +0400
```

```
+++ ./mm/rmap.c 2006-09-05 13:51:55.000000000 +0400
```

```
@@ -54,6 +54,8 @@
```

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

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

```
+#include <bc/vmrss.h>
```

```
+
```

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

```
struct kmem_cache *anon_vma_cache;
```

```
@@ -687,6 +689,7 @@ static int try_to_unmap_one(struct page
    dec_mm_counter(mm, file_rss);
```

```
+ bc_vmrss_page_del(page, mm, vma);
  page_remove_rmap(page);
  page_cache_release(page);
```

```
@@ -777,6 +780,7 @@ static void try_to_unmap_cluster(unsigne
    if (pte_dirty(pteval))
        set_page_dirty(page);
```

```
+ bc_vmrss_page_del(page, mm, vma);
  page_remove_rmap(page);
  page_cache_release(page);
  dec_mm_counter(mm, file_rss);
```

```
--- ./mm/swapfile.c.bcrssch 2006-09-05 12:53:59.000000000 +0400
```

```
+++ ./mm/swapfile.c 2006-09-05 13:54:59.000000000 +0400
```

```

@@ -28,6 +28,9 @@
#include <linux/capability.h>
#include <linux/syscalls.h>

+#include <bc/beancounter.h>
+#include <bc/vmrss.h>
+
#include <asm/pgtable.h>
#include <asm/tlbflush.h>
#include <linux/swapops.h>
@@ -487,13 +490,15 @@ 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_beancounter **ppb)
{
    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);
+ bc_vmrss_page_add(page, vma->vm_mm, vma, ppb);
    swap_free(entry);
    /*
     * Move the page to the active list so it is not
@@ -504,7 +509,8 @@ static void unuse_pte(struct vm_area_str

static int unuse_pte_range(struct vm_area_struct *vma, pmd_t *pmd,
    unsigned long addr, unsigned long end,
- swp_entry_t entry, struct page *page)
+ swp_entry_t entry, struct page *page,
+ struct page_beancounter **ppb)
{
    pte_t swp_pte = swp_entry_to_pte(entry);
    pte_t *pte;
@@ -518,7 +524,7 @@ 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, ppb);
        found = 1;
        break;
    }
}
@@ -529,7 +535,8 @@ static int unuse_pte_range(struct vm_are

```

```

static inline int unuse_pmd_range(struct vm_area_struct *vma, pud_t *pud,
    unsigned long addr, unsigned long end,
-   swp_entry_t entry, struct page *page)
+   swp_entry_t entry, struct page *page,
+   struct page_beancounter **ppb)
{
    pmd_t *pmd;
    unsigned long next;
@@ -539,7 +546,7 @@ static inline int unuse_pmd_range(struct
    next = pmd_addr_end(addr, end);
    if (pmd_none_or_clear_bad(pmd))
        continue;
-   if (unuse_pte_range(vma, pmd, addr, next, entry, page))
+   if (unuse_pte_range(vma, pmd, addr, next, entry, page, ppb))
        return 1;
    } while (pmd++, addr = next, addr != end);
    return 0;
@@ -547,7 +554,8 @@ static inline int unuse_pmd_range(struct

static inline int unuse_pud_range(struct vm_area_struct *vma, pgd_t *pgd,
    unsigned long addr, unsigned long end,
-   swp_entry_t entry, struct page *page)
+   swp_entry_t entry, struct page *page,
+   struct page_beancounter **ppb)
{
    pud_t *pud;
    unsigned long next;
@@ -557,14 +565,15 @@ static inline int unuse_pud_range(struct
    next = pud_addr_end(addr, end);
    if (pud_none_or_clear_bad(pud))
        continue;
-   if (unuse_pmd_range(vma, pud, addr, next, entry, page))
+   if (unuse_pmd_range(vma, pud, addr, next, entry, page, ppb))
        return 1;
    } while (pud++, addr = next, addr != end);
    return 0;
}

static int unuse_vma(struct vm_area_struct *vma,
-   swp_entry_t entry, struct page *page)
+   swp_entry_t entry, struct page *page,
+   struct page_beancounter **ppb)
{
    pgd_t *pgd;
    unsigned long addr, end, next;
@@ -585,14 +594,15 @@ 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))
+ if (unuse_pud_range(vma, pgd, addr, next, entry, page, ppb))
    return 1;
} while (pgd++, addr = next, addr != end);
return 0;
}

static int unuse_mm(struct mm_struct *mm,
-   swp_entry_t entry, struct page *page)
+   swp_entry_t entry, struct page *page,
+   struct page_beancounter **ppb)
{
    struct vm_area_struct *vma;

@@ -607,7 +617,7 @@ 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 && unuse_vma(vma, entry, page, ppb))
    break;
}
up_read(&mm->mmap_sem);
@@ -673,6 +683,7 @@ static int try_to_unuse(unsigned int typ
    int retval = 0;
    int reset_overflow = 0;
    int shmem;
+ struct page_beancounter *pb;

    /*
     * When searching mms for an entry, a good strategy is to
@@ -692,6 +703,7 @@ static int try_to_unuse(unsigned int typ
    start_mm = &init_mm;
    atomic_inc(&init_mm.mm_users);

+ pb = NULL;
    /*
     * Keep on scanning until all entries have gone. Usually,
     * one pass through swap_map is enough, but not necessarily:
@@ -703,6 +715,12 @@ static int try_to_unuse(unsigned int typ
    break;
}

+ pb = bc_alloc_rss_counter_list(nr_beancounters, pb);
+ if (IS_ERR(pb)) {
+   retval = PTR_ERR(pb);
+   break;

```

```

+ }
+
+ /*
+  * Get a page for the entry, using the existing swap
+  * cache page if there is one. Otherwise, get a clean
@@ -757,7 +775,7 @@ static int try_to_unuse(unsigned int typ
    if (start_mm == &init_mm)
        shmem = shmem_unuse(entry, page);
    else
-   retval = unuse_mm(start_mm, entry, page);
+   retval = unuse_mm(start_mm, entry, page, &pb);
    }
    if (*swap_map > 1) {
        int set_start_mm = (*swap_map >= swcount);
@@ -787,7 +805,7 @@ static int try_to_unuse(unsigned int typ
        set_start_mm = 1;
        shmem = shmem_unuse(entry, page);
    } else
-   retval = unuse_mm(mm, entry, page);
+   retval = unuse_mm(mm, entry, page, &pb);
    if (set_start_mm && *swap_map < swcount) {
        mmput(new_start_mm);
        atomic_inc(&mm->mm_users);
@@ -878,6 +896,9 @@ static int try_to_unuse(unsigned int typ
        cond_resched();
    }

+   if (!IS_ERR(pb))
+       bc_free_rss_counter(pb);
+
    mmput(start_mm);
    if (reset_overflow) {
        printk(KERN_WARNING "swapoff: cleared swap entry overflow\n");

```
