Subject: [PATCH 12/13] BC: vmrss (core)
Posted by dev on Tue, 05 Sep 2006 15:28:58 GMT
View Forum Message <> Reply to Message

This is the core of vmrss accounting.

The main introduced object is page_beancounter.
It ties together page and BCs which use the page.
This allows correctly account fractions of memory shared
between BCs (http://wiki.openvz.org/RSS_fractions_accounting)

Accounting API:
1. bc_alloc_rss_counter() allocates a tie between page and BC
2. bc_free_rss_counter frees it.

(1) and (2) must be done each time a page is about
to be added to someone's rss.

3. When page is touched by BC (i.e. by any task which mm belongs to BC)
   page is bc_vmrss_page_add()-ed to that BC. Touching page leads
   to subtracting it from unused_prvvmpages and adding to held_pages.
4. When page is unmapped from BC it is bc_vmrss_page_del()-ed from it.

5. When task forks all it's mapped pages must be bc_vmrss_page_dup()-ed.
   i.e. page beancounter reference counter must be increased.

6. Some pages (former PGReserved) must be added to rss, but without
   having a reference on it. These pages are bc_vmrss_page_add_noref()-ed.

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

```
 include/bc/beancounter.h |    3
 include/bc/vmpages.h     |    4
 include/bc/vmrss.h       |   72 ++++++
 include/linux/mm.h       |    6
 include/linux/shmem_fs.h |    2
 init/main.c              |    2
 kernel/bc/Kconfig        |    9
 kernel/bc/Makefile       |    1
 kernel/bc/beancounter.c  |    9
 kernel/bc/vmpages.c      |    7
 kernel/bc/vmrss.c        |  508 +++++++++++++++++++++++++++++++++++++++++++++++++
 mm/shmem.c               |    6
 12 files changed, 627 insertions(+), 2 deletions(-)
```

```
--- ./include/bc/beancounter.h.bcrsscore 2006-09-05 13:44:33.000000000 +0400
+++ ./include/bc/beancounter.h 2006-09-05 13:50:29.000000000 +0400
@@ -68,6 +68,9 @@ struct beancounter {
  struct hlist_node hash;

  unsigned long  unused_privvmpages;
+#ifdef CONFIG_BEANCOUNTERS_RSS
+ unsigned long long rss_pages;
+#endif
  /* resources statistics and settings */
  struct bc_resource_parm bc_parms[BC_RESOURCES];
 };
--- ./include/bc/vmpages.h.bcrsscore 2006-09-05 13:40:21.000000000 +0400
+++ ./include/bc/vmpages.h 2006-09-05 13:46:35.000000000 +0400
@@ -77,6 +77,8 @@ void bc_locked_shm_uncharge(struct shmem
   put_beancounter((info)->shm_bc);   \
  } while (0)

+#define mm_same_bc(mm1, mm2) ((mm1)->mm_bc == (mm2)->mm_bc)
+
 void bc_update_privvmpages(struct beancounter *bc);

 #else /* CONFIG_BEANCOUNTERS */
@@ -136,6 +138,8 @@ static inline void bc_locked_shm_uncharg
 #define shmi_init_bc(info) do { } while (0)
 #define shmi_free_bc(info) do { } while (0)

+#define mm_same_bc(mm1, mm2) (1)
+
 #endif /* CONFIG_BEANCOUNTERS */
 #endif

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/vmrss.h 2006-09-05 13:50:25.000000000 +0400
@@ -0,0 +1,72 @@
+/*
+ *  include/ub/vmrss.h
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#ifndef __BC_VMRSS_H_
+#define __BC_VMRSS_H_
+
+struct page_beancounter;
+
+struct page;
```

```
+struct mm_struct;
+struct vm_area_struct;
+
+/* values that represens page's 'weight' in bc rss accounting */
+#define PB_PAGE_WEIGHT_SHIFT 24
+#define PB_PAGE_WEIGHT (1 << PB_PAGE_WEIGHT_SHIFT)
+/* page obtains one more reference within beancounter */
+#define PB_COPY_SAME ((struct page_beancounter *)-1)
+
+#ifdef CONFIG_BEANCOUNTERS_RSS
+
+struct page_beancounter * __must_check bc_alloc_rss_counter(void);
+struct page_beancounter * __must_check bc_alloc_rss_counter_list(long num,
+  struct page_beancounter *list);
+
+void bc_free_rss_counter(struct page_beancounter *rc);
+
+void bc_vmrss_page_add(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma, struct page_beancounter **ppb);
+void bc_vmrss_page_del(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma);
+void bc_vmrss_page_dup(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma, struct page_beancounter **ppb);
+void bc_vmrss_page_add_noref(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma);
+
+unsigned long mm_rss_pages(struct mm_struct *mm, unsigned long start,
+  unsigned long end);
+
+void bc_init_rss(void);
+
+#else /* CONFIG_BEANCOUNTERS_RSS */
+
+static inline struct page_beancounter * __must_check bc_alloc_rss_counter(void)
+{
+ return NULL;
+}
+
+static inline struct page_beancounter * __must_check bc_alloc_rss_counter_list(
+  long num, struct page_beancounter *list)
+{
+ return NULL;
+}
+
+static inline void bc_free_rss_counter(struct page_beancounter *rc)
+{
+}
+
```

```
+#define bc_vmrss_page_add(pg, mm, vma, pb) do { } while (0)
+#define bc_vmrss_page_del(pg, mm, vma)  do { } while (0)
+#define bc_vmrss_page_dup(pg, mm, vma, pb) do { } while (0)
+#define bc_vmrss_page_add_noref(pg, mm, vma) do { } while (0)
+#define mm_rss_pages(mm, start, end) (0)
+
+#define bc_init_rss()   do { } while (0)
+
+#endif /* CONFIG_BEANCOUNTERS_RSS */
+
+#endif
--- ./include/linux/mm.h.bcrsscore 2006-09-05 13:06:37.000000000 +0400
+++ ./include/linux/mm.h 2006-09-05 13:47:12.000000000 +0400
@@ -275,11 +275,15 @@ struct page {
  unsigned long trace[8];
 #endif
 #ifdef CONFIG_BEANCOUNTERS
- struct beancounter *page_bc;
+ union {
+  struct beancounter *page_bc;
+  struct page_beancounter *page_pb;
+ };
 #endif
 };

 #define page_bc(page)   ((page)->page_bc)
+#define page_pb(page)   ((page)->page_pb)
 #define page_private(page)  ((page)->private)
 #define set_page_private(page, v) ((page)->private = (v))

--- ./include/linux/shmem_fs.h.bcrsscore 2006-09-05 12:59:27.000000000 +0400
+++ ./include/linux/shmem_fs.h 2006-09-05 13:50:19.000000000 +0400
@@ -41,4 +41,6 @@ static inline struct shmem_inode_info *S
  return container_of(inode, struct shmem_inode_info, vfs_inode);
 }

+int is_shmem_mapping(struct address_space *mapping);
+
 #endif
--- ./init/main.c.bcrsscore 2006-09-05 12:54:17.000000000 +0400
+++ ./init/main.c 2006-09-05 13:46:35.000000000 +0400
@@ -51,6 +51,7 @@
 #include <linux/lockdep.h>

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

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

```
 #include <asm/bugs.h>
@@ -608,6 +609,7 @@ asmlinkage void __init start_kernel(void
  check_bugs();

  acpi_early_init(); /* before LAPIC and SMP init */
+ bc_init_rss();

  /* Do the rest non-__init'ed, we're now alive */
  rest_init();
--- ./kernel/bc/Kconfig.bcrsscore 2006-09-05 12:54:14.000000000 +0400
+++ ./kernel/bc/Kconfig 2006-09-05 13:50:35.000000000 +0400
@@ -22,4 +22,13 @@ config BEANCOUNTERS
          per-process basis.  Per-process accounting doesn't prevent malicious
          users from spawning a lot of resource-consuming processes.

+config BEANCOUNTERS_RSS
+ bool "Account physical memory usage"
+ default y
+ depends on BEANCOUNTERS
+ help
+   This allows to estimate per beancounter physical memory usage.
+   Implemented alghorithm accounts shared pages of memory as well,
+   dividing them by number of beancounter which use the page.
+
 endmenu
--- ./kernel/bc/Makefile.bcrsscore 2006-09-05 12:59:37.000000000 +0400
+++ ./kernel/bc/Makefile 2006-09-05 13:50:48.000000000 +0400
@@ -9,3 +9,4 @@ obj-y += misc.o
 obj-y += sys.o
 obj-y += kmem.o
 obj-y += vmpages.o
+obj-$(CONFIG_BEANCOUNTERS_RSS) += vmrss.o
--- ./kernel/bc/beancounter.c.bcrsscore 2006-09-05 13:44:53.000000000 +0400
+++ ./kernel/bc/beancounter.c 2006-09-05 13:49:38.000000000 +0400
@@ -11,6 +11,7 @@
 #include <linux/hash.h>

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

 static kmem_cache_t *bc_cachep;
 static struct beancounter default_beancounter;
@@ -112,6 +113,14 @@ void put_beancounter(struct beancounter
    printk("BC: %d has %lu of %s held on put", bc->bc_id,
     bc->bc_parms[i].held, bc_rnames[i]);

+ if (bc->unused_privvmpages != 0)
+  printk("BC: %d has %lu of unused pages held on put", bc->bc_id,
```

```
+   bc->unused_privvmpages);
+#ifdef CONFIG_BEANCOUNTERS_RSS
+ if (bc->rss_pages != 0)
+  printk("BC: %d hash %llu of rss pages held on put", bc->bc_id,
+   bc->rss_pages);
+#endif
  hlist_del(&bc->hash);
  nr_beancounters--;
  spin_unlock_irqrestore(&bc_hash_lock, flags);
--- ./kernel/bc/vmpages.c.bcrsscore 2006-09-05 13:45:34.000000000 +0400
+++ ./kernel/bc/vmpages.c 2006-09-05 13:48:50.000000000 +0400
@@ -11,12 +11,17 @@

 #include <bc/beancounter.h>
 #include <bc/vmpages.h>
+#include <bc/vmrss.h>

 #include <asm/page.h>

 void bc_update_privvmpages(struct beancounter *bc)
 {
- bc->bc_parms[BC_PRIVVMPAGES].held = bc->unused_privvmpages;
+ bc->bc_parms[BC_PRIVVMPAGES].held = bc->unused_privvmpages
+#ifdef CONFIG_BEANCOUNTERS_RSS
+  + (bc->rss_pages >> PB_PAGE_WEIGHT_SHIFT)
+#endif
+  ;
  bc_adjust_minheld(bc, BC_PRIVVMPAGES);
  bc_adjust_maxheld(bc, BC_PRIVVMPAGES);
 }
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/vmrss.c 2006-09-05 13:51:21.000000000 +0400
@@ -0,0 +1,508 @@
+/*
+ *  kernel/bc/vmrss.c
+ *
+ *  Copyright (C) 2006 OpenVZ. SWsoft Inc
+ *
+ */
+
+#include <linux/sched.h>
+#include <linux/mm.h>
+#include <linux/list.h>
+#include <linux/slab.h>
+#include <linux/vmalloc.h>
+#include <linux/shmem_fs.h>
+#include <linux/highmem.h>
+
```

```
+#include <bc/beancounter.h>
+#include <bc/vmpages.h>
+#include <bc/vmrss.h>
+
+#include <asm/pgtable.h>
+
+/*
+ * Core object of accounting.
+ * page_beancounter (or rss_counter) ties together page an bc.
+ * Page has associated circular list of such pbs. When page is
+ * shared between bcs then it's size is splitted between all of
+ * them in 2^n-s parts.
+ *
+ * E.g. three bcs will share page like 1/2:1/4:1/4
+ * adding one more reference would produce such a change:
+ * 1/2(bc1) : 1/4(bc2) : 1/4(bc3) ->
+ * (1/4(bc1) + 1/4(bc1)) : 1/4(bc2) : 1/4(bc3) ->
+ * 1/4(bc2) : 1/4(bc3) : 1/4(bc4) : 1/4(bc1)
+ */
+
+#define PB_MAGIC 0x62700001UL
+
+struct page_beancounter {
+ unsigned long magic;
+ struct page *page;
+ struct beancounter *bc;
+ struct page_beancounter *next_hash;
+ unsigned refcount;
+ struct list_head page_list;
+};
+
+#define PB_REFC_BITS 24
+
+#define pb_shift(p) ((p)->refcount >> PB_REFC_BITS)
+#define pb_shift_inc(p) do { ((p)->refcount += (1 << PB_REFC_BITS)); } while (0)
+#define pb_shift_dec(p) do { ((p)->refcount -= (1 << PB_REFC_BITS)); } while (0)
+
+#define pb_count(p) ((p)->refcount & ((1 << PB_REFC_BITS) - 1))
+#define pb_get(p) do { ((p)->refcount++); } while (0)
+#define pb_put(p) do { ((p)->refcount--); } while (0)
+
+#define pb_refcount_init(p, shift) do {     \
+  (p)->refcount = ((shift) << PB_REFC_BITS) + (1); \
+ } while (0)
+
+static spinlock_t pb_lock = SPIN_LOCK_UNLOCKED;
+static struct page_beancounter **pb_hash_table;
+static unsigned int pb_hash_mask;
```

```
+
+static inline int pb_hash(struct beancounter *bc, struct page *page)
+{
+ return (page_to_pfn(page) + (bc->bc_id << 10)) & pb_hash_mask;
+}
+
+static kmem_cache_t *pb_cachep;
+#define alloc_pb() kmem_cache_alloc(pb_cachep, GFP_KERNEL)
+#define free_pb(p) kmem_cache_free(pb_cachep, p)
+
+#define next_page_pb(p) list_entry(p->page_list.next, \
+  struct page_beancounter, page_list);
+#define prev_page_pb(p) list_entry(p->page_list.prev, \
+  struct page_beancounter, page_list);
+
+/*
+ * Allocates a new page_beancounter struct and
+ * initialises requred fields.
+ * pb->next_hash is set to NULL as this field is used
+ * in two ways:
+ * 1. When pb is in hash - it points to the next one in
+ *    the current hash chain;
+ * 2. When pb is not in hash yet - it points to the next pb
+ *    in list just allocated.
+ */
+struct page_beancounter *bc_alloc_rss_counter(void)
+{
+ struct page_beancounter *pb;
+
+ pb = alloc_pb();
+ if (pb == NULL)
+  return ERR_PTR(-ENOMEM);
+
+ pb->magic = PB_MAGIC;
+ pb->next_hash = NULL;
+ return pb;
+}
+
+/*
+ * This function ensures that @list has at least @num elements.
+ * Otherwise needed elements are allocated and new list is
+ * returned. On error old list is freed.
+ *
+ * num == BC_ALLOC_ALL means that lis must contain as many
+ * elements as there are BCCs in hash now.
+ */
+struct page_beancounter *bc_alloc_rss_counter_list(long num,
+  struct page_beancounter *list)
```

```
+{
+ struct page_beancounter *pb;
+
+ for (pb = list; pb != NULL && num != 0; pb = pb->next_hash, num--);
+
+ /* need to allocate num more elements */
+ while (num > 0) {
+  pb = alloc_pb();
+  if (pb == NULL)
+   goto err;
+
+  pb->magic = PB_MAGIC;
+  pb->next_hash = list;
+  list = pb;
+  num--;
+ }
+
+ return list;
+
+err:
+ bc_free_rss_counter(list);
+ return ERR_PTR(-ENOMEM);
+}
+
+/*
+ * Free the list of page_beancounter-s
+ */
+void bc_free_rss_counter(struct page_beancounter *pb)
+{
+ struct page_beancounter *tmp;
+
+ while (pb) {
+  tmp = pb->next_hash;
+  free_pb(pb);
+  pb = tmp;
+ }
+}
+
+/*
+ * Helpers to update rss_pages and unused_privvmpages on BC
+ */
+static void mod_rss_pages(struct beancounter *bc, int val,
+  struct vm_area_struct *vma, int unused)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&bc->bc_lock, flags);
+ if (vma && BC_VM_PRIVATE(vma->vm_flags, vma->vm_file)) {
```

```
+  if (unused < 0 && unlikely(bc->unused_privvmpages < -unused)) {
+   printk("BC: overuncharging %d unused pages: "
+     "val %i, held %lu\n",
+     bc->bc_id, unused,
+     bc->unused_privvmpages);
+   unused = -bc->unused_privvmpages;
+  }
+  bc->unused_privvmpages += unused;
+ }
+ bc->rss_pages += val;
+ bc_update_privvmpages(bc);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+
+#define __inc_rss_pages(bc, val) mod_rss_pages(bc, val, NULL, 0)
+#define __dec_rss_pages(bc, val) mod_rss_pages(bc, -(val), NULL, 0)
+#define inc_rss_pages(bc, val, vma) mod_rss_pages(bc, val, vma, -1)
+#define dec_rss_pages(bc, val, vma) mod_rss_pages(bc, -(val), vma, 1)
+
+/*
+ * Routines to manipulate page-to-bc references (page_beancounter)
+ * Reference may be added, removed or duplicated (see descriptions below)
+ */
+
+static int __pb_dup_ref(struct page *pg, struct beancounter *bc, int hash)
+{
+ struct page_beancounter *p;
+
+ for (p = pb_hash_table[hash];
+    p != NULL && (p->page != pg || p->bc != bc);
+    p = p->next_hash);
+ if (p == NULL)
+  return -1;
+
+ pb_get(p);
+ return 0;
+}
+
+static int __pb_add_ref(struct page *pg, struct beancounter *bc,
+  int hash, struct page_beancounter **ppb)
+{
+ struct page_beancounter *head, *p;
+ int shift, ret;
+
+ p = *ppb;
+ *ppb = p->next_hash;
+
+ p->page = pg;
```

```
+ p->bc = get_beancounter(bc);
+ p->next_hash = pb_hash_table[hash];
+ pb_hash_table[hash] = p;
+
+ head = page_pb(pg);
+ if (head != NULL) {
+  BUG_ON(head->magic != PB_MAGIC);
+  /*
+   * Move the first element to the end of the list.
+   * List head (pb_head) is set to the next entry.
+   * Note that this code works even if head is the only element
+   * on the list (because it's cyclic).
+   */
+  page_pb(pg) = next_page_pb(head);
+  pb_shift_inc(head);
+  shift = pb_shift(head);
+  /*
+   * Update user beancounter, the share of head has been changed.
+   * Note that the shift counter is taken after increment.
+   */
+  __dec_rss_pages(head->bc, PB_PAGE_WEIGHT >> shift);
+  /*
+   * Add the new page beancounter to the end of the list.
+   */
+  list_add_tail(&p->page_list, &page_pb(pg)->page_list);
+ } else {
+  page_pb(pg) = p;
+  shift = 0;
+  INIT_LIST_HEAD(&p->page_list);
+ }
+
+ pb_refcount_init(p, shift);
+ ret = PB_PAGE_WEIGHT >> shift;
+ return ret;
+}
+
+static int __pb_remove_ref(struct page *page, struct beancounter *bc)
+{
+ int hash, ret;
+ struct page_beancounter *p, **q;
+ int shift, shiftt;
+
+ ret = 0;
+
+ hash = pb_hash(bc, page);
+
+ BUG_ON(page_pb(page) != NULL && page_pb(page)->magic != PB_MAGIC);
+ for (q = pb_hash_table + hash, p = *q;
```

```
+   p != NULL && (p->page != page || p->bc != bc);
+   q = &p->next_hash, p = *q);
+ if (p == NULL)
+  goto out;
+
+ pb_put(p);
+ if (pb_count(p) > 0)
+  goto out;
+
+ /* remove from the hash list */
+ *q = p->next_hash;
+
+ shift = pb_shift(p);
+ ret = PB_PAGE_WEIGHT >> shift;
+
+ if (page_pb(page) == p) {
+  if (list_empty(&p->page_list)) {
+   page_pb(page) = NULL;
+   put_beancounter(bc);
+   free_pb(p);
+   goto out;
+  }
+  page_pb(page) = next_page_pb(p);
+ }
+
+ list_del(&p->page_list);
+ put_beancounter(bc);
+ free_pb(p);
+
+ /*
+  * Now balance the list.
+  * Move the tail and adjust its shift counter.
+  */
+ p = prev_page_pb(page_pb(page));
+ shiftt = pb_shift(p);
+ pb_shift_dec(p);
+ page_pb(page) = p;
+ __inc_rss_pages(p->bc, PB_PAGE_WEIGHT >> shiftt);
+
+ /*
+  * If the shift counter of the moved beancounter is different from the
+  * removed one's, repeat the procedure for one more tail beancounter
+  */
+ if (shiftt > shift) {
+  p = prev_page_pb(page_pb(page));
+  pb_shift_dec(p);
+  page_pb(page) = p;
+  __inc_rss_pages(p->bc, PB_PAGE_WEIGHT >> shiftt);
```

```
+ }
+out:
+ return ret;
+}
+
+/*
+ * bc_vmrss_page_add: Called when page is added to resident set
+ *   of any mm. In this case page is substracted from unused_privvmpages
+ *   (if it is BC_VM_PRIVATE one) and a reference to BC must be set
+ *   with page_beancounter.
+ *
+ * bc_vmrss_page_del: The reverse operation - page is removed from
+ *   resident set and must become unused.
+ *
+ * bc_vmrss_page_dup: This is called on dup_mmap() when all pages
+ *   become shared between two mm structs. This case has one feature:
+ *   some pages (see below) may lack a reference to BC, so setting
+ *   new reference is not needed, but update of unused_privvmpages
+ *   is required.
+ *
+ * bc_vmrss_page_add_noref: This is called for (former) reserved pages
+ *   like ZERO_PAGE() or some pages set up with insert_page(). These
+ *   pages must not have reference to any BC, but must be accounted in
+ *   rss.
+ */
+
+void bc_vmrss_page_add(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma, struct page_beancounter **ppb)
+{
+ struct beancounter *bc;
+ int hash, ret;
+
+ if (!PageAnon(pg) && is_shmem_mapping(pg->mapping))
+  return;
+
+ bc = mm->mm_bc;
+ hash = pb_hash(bc, pg);
+
+ ret = 0;
+ spin_lock(&pb_lock);
+ if (__pb_dup_ref(pg, bc, hash))
+  ret = __pb_add_ref(pg, bc, hash, ppb);
+ spin_unlock(&pb_lock);
+
+ inc_rss_pages(bc, ret, vma);
+}
+
+void bc_vmrss_page_del(struct page *pg, struct mm_struct *mm,
```

```
+  struct vm_area_struct *vma)
+{
+ struct beancounter *bc;
+ int ret;
+
+ if (!PageAnon(pg) && is_shmem_mapping(pg->mapping))
+  return;
+
+ bc = mm->mm_bc;
+
+ spin_lock(&pb_lock);
+ ret = __pb_remove_ref(pg, bc);
+ spin_unlock(&pb_lock);
+
+ dec_rss_pages(bc, ret, vma);
+}
+
+void bc_vmrss_page_dup(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma, struct page_beancounter **ppb)
+{
+ struct beancounter *bc;
+ int hash, ret;
+
+ if (!PageAnon(pg) && is_shmem_mapping(pg->mapping))
+  return;
+
+ bc = mm->mm_bc;
+ hash = pb_hash(bc, pg);
+
+ ret = 0;
+ spin_lock(&pb_lock);
+ if (page_pb(pg) == NULL)
+  /*
+   * pages like ZERO_PAGE must not be accounted in pbc
+   * so on fork we just skip them
+   */
+  goto out_unlock;
+
+ if (*ppb == PB_COPY_SAME) {
+  if (__pb_dup_ref(pg, bc, hash))
+   WARN_ON(1);
+ } else
+  ret = __pb_add_ref(pg, bc, hash, ppb);
+out_unlock:
+ spin_unlock(&pb_lock);
+
+ inc_rss_pages(bc, ret, vma);
+}
```

```
+
+void bc_vmrss_page_add_noref(struct page *pg, struct mm_struct *mm,
+  struct vm_area_struct *vma)
+{
+ inc_rss_pages(mm->mm_bc, 0, vma);
+}
+
+/*
+ * Calculate the number of currently resident pages for
+ * given mm_struct in a given range (addr - end).
+ * This is needed for mprotect_fixup() as by the time
+ * it is called some pages can be resident and thus
+ * not accounted in bc->unused_privvmpages. Such pages
+ * must num be uncharged (as they already are).
+ */
+
+static unsigned long pages_in_pte_range(struct mm_struct *mm, pmd_t *pmd,
+    unsigned long addr, unsigned long end,
+    unsigned long *pages)
+{
+ pte_t *pte;
+ spinlock_t *ptl;
+
+ pte = pte_offset_map_lock(mm, pmd, addr, &ptl);
+ do {
+  pte_t ptent = *pte;
+  if (!pte_none(ptent) && pte_present(ptent))
+   (*pages)++;
+ } while (pte++, addr += PAGE_SIZE, addr != end);
+ pte_unmap_unlock(pte - 1, ptl);
+ return addr;
+}
+
+static inline unsigned long pages_in_pmd_range(struct mm_struct *mm, pud_t *pud,
+    unsigned long addr, unsigned long end,
+    unsigned long *pages)
+{
+ pmd_t *pmd;
+ unsigned long next;
+
+ pmd = pmd_offset(pud, addr);
+ do {
+  next = pmd_addr_end(addr, end);
+  if (pmd_none_or_clear_bad(pmd))
+   continue;
+
+  next = pages_in_pte_range(mm, pmd, addr, next, pages);
+ } while (pmd++, addr = next, addr != end);
```

```
+ return addr;
+}
+
+static inline unsigned long pages_in_pud_range(struct mm_struct *mm, pgd_t *pgd,
+    unsigned long addr, unsigned long end,
+    unsigned long *pages)
+{
+ pud_t *pud;
+ unsigned long next;
+
+ pud = pud_offset(pgd, addr);
+ do {
+  next = pud_addr_end(addr, end);
+  if (pud_none_or_clear_bad(pud))
+   continue;
+
+  next = pages_in_pmd_range(mm, pud, addr, next, pages);
+ } while (pud++, addr = next, addr != end);
+ return addr;
+}
+
+unsigned long mm_rss_pages(struct mm_struct *mm,
+  unsigned long addr, unsigned long end)
+{
+ pgd_t *pgd;
+ unsigned long next;
+ unsigned long pages;
+
+ BUG_ON(addr >= end);
+
+ pages = 0;
+ pgd = pgd_offset(mm, addr);
+ do {
+  next = pgd_addr_end(addr, end);
+  if (pgd_none_or_clear_bad(pgd))
+   continue;
+
+  next = pages_in_pud_range(mm, pgd, addr, next, &pages);
+ } while (pgd++, addr = next, addr != end);
+ return pages;
+}
+
+void __init bc_init_rss(void)
+{
+ unsigned long hash_size;
+
+ pb_cachep = kmem_cache_create("page_beancounter",
+   sizeof(struct page_beancounter), 0,
```

```
+   SLAB_HWCACHE_ALIGN | SLAB_PANIC, NULL, NULL);
+
+ hash_size = num_physpages >> 2;
+ for (pb_hash_mask = 1;
+   (hash_size & pb_hash_mask) != hash_size;
+   pb_hash_mask = (pb_hash_mask << 1) + 1);
+
+ hash_size = pb_hash_mask + 1;
+ printk(KERN_INFO "BC: Page beancounter hash is %lu entries.\n",
+   hash_size);
+ pb_hash_table = vmalloc(hash_size * sizeof(struct page_beancounter *));
+ memset(pb_hash_table, 0, hash_size * sizeof(struct page_beancounter *));
+}
--- ./mm/shmem.c.bcrsscore 2006-09-05 13:39:26.000000000 +0400
+++ ./mm/shmem.c 2006-09-05 13:46:35.000000000 +0400
@@ -2236,6 +2236,12 @@ static struct vm_operations_struct shmem
 #endif
 };

+#ifdef CONFIG_BEANCOUNTERS_RSS
+int is_shmem_mapping(struct address_space *mapping)
+{
+ return (mapping != NULL && mapping->a_ops == &shmem_aops);
+}
+#endif

 static int shmem_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
```