Subject: [PATCH 10/13] BC: physpages accounting (core)
Posted by dev on Thu, 09 Nov 2006 16:59:40 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.
page_beancounter also allows quick per-container pages reclamation.
and helps correctly account fractions of memory shared
between BCs (http://wiki.openvz.org/RSS_fractions_accounting)

Page charge/uncharge is performed on first map/last unmap
and is based on page->mapcount calculations.

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

```
 include/bc/beancounter.h |    2
 include/bc/rsspages.h    |   46 ++++++++
 include/linux/mm.h       |    5
 include/linux/mm_types.h |    5
 kernel/bc/beancounter.c  |    2
 kernel/bc/rsspages.c     |  267 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 6 files changed, 326 insertions(+), 1 deletion(-)

--- ./include/bc/beancounter.h.ve9 2006-11-07 12:03:41.000000000 +0300
+++ ./include/bc/beancounter.h 2006-11-07 12:03:47.000000000 +0300
@@ -67,6 +67,8 @@ struct beancounter {
  bcid_t   bc_id;
  struct hlist_node bc_hash;

+ spinlock_t  bc_page_lock;
+ struct list_head bc_page_list;
  struct bc_resource_parm bc_parms[BC_RESOURCES];
 };

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/rsspages.h 2006-11-07 12:03:57.000000000 +0300
@@ -0,0 +1,46 @@
+/*
+ * include/bc/rsspages.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
```

```
+
+#ifndef __BC_RSSPAGES_H_
+#define __BC_RSSPAGES_H_
+
+#include <linux/compiler.h>
+
+struct page;
+struct vm_area_struct;
+struct page_beancounter;
+
+#ifdef CONFIG_BEANCOUNTERS
+int __must_check bc_rsspage_prepare(struct page *p,
+  struct vm_area_struct *vma, struct page_beancounter **ppb);
+void bc_rsspage_charge(struct page_beancounter *pb);
+void bc_rsspage_release(struct page_beancounter *pb);
+void bc_rsspage_uncharge(struct page_beancounter *pb);
+
+unsigned long bc_try_to_free_pages(struct beancounter *bc);
+unsigned long bc_isolate_pages(unsigned long nr_to_scan,
+  struct beancounter *bc, struct list_head *dst,
+  int active, unsigned long *scanned);
+unsigned long bc_nr_physpages(struct beancounter *bc);
+#else
+static inline int __must_check bc_rsspage_prepare(struct page *p,
+  struct vm_area_struct *vma, struct page_beancounter **ppb)
+{
+ return 0;
+}
+
+static inline void bc_rsspage_charge(struct page_beancounter *pb)
+{
+}
+static inline void bc_rsspage_release(struct page_beancounter *pb)
+{
+}
+static inline void bc_rsspage_uncharge(struct page_beancounter *pb)
+{
+}
+#endif
+#endif
--- ./include/linux/mm.h.ve9 2006-11-07 12:03:41.000000000 +0300
+++ ./include/linux/mm.h 2006-11-07 12:03:47.000000000 +0300
@@ -223,6 +223,11 @@ struct mmu_gather;
 struct inode;

 #define page_bc(page)   ((page)->bc)
+#ifdef CONFIG_BEANCOUNTERS
+#define page_pb(page)   ((page)->pb)
```

```
+#else
+#define page_pb(page)   (NULL)
+#endif
 #define page_private(page)  ((page)->private)
 #define set_page_private(page, v) ((page)->private = (v))

--- ./include/linux/mm_types.h.ve9 2006-11-07 12:03:41.000000000 +0300
+++ ./include/linux/mm_types.h 2006-11-07 12:03:47.000000000 +0300
@@ -63,7 +63,10 @@ struct page {
         not kmapped, ie. highmem) */
 #endif /* WANT_PAGE_VIRTUAL */
 #ifdef CONFIG_BEANCOUNTERS
- struct beancounter *bc;
+ union {
+  struct beancounter *bc;
+  struct page_beancounter *pb;
+ };
 #endif
 #ifdef CONFIG_PAGE_OWNER
   int order;
--- ./kernel/bc/beancounter.c.ve9 2006-11-07 12:03:41.000000000 +0300
+++ ./kernel/bc/beancounter.c 2006-11-07 12:03:47.000000000 +0300
@@ -227,6 +227,8 @@ void __init bc_init_early(void)
   int i;

   init_beancounter_struct(&init_bc, 0);
+ spin_lock_init(&init_bc.bc_page_lock);
+ INIT_LIST_HEAD(&init_bc.bc_page_list);

   for (i = 0; i < BC_RESOURCES; i++) {
    init_bc.bc_parms[i].barrier = BC_MAXVALUE;
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/rsspages.c 2006-11-07 12:03:47.000000000 +0300
@@ -0,0 +1,267 @@
+/*
+ * kernel/bc/rsspages.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#include <linux/mm_types.h>
+#include <linux/mm.h>
+#include <linux/page-flags.h>
+#include <linux/hardirq.h>
+#include <linux/kernel.h>
+
+#include <bc/beancounter.h>
```

```
+#include <bc/vmpages.h>
+#include <bc/rsspages.h>
+
+#include <asm/bitops.h>
+
+#define BC_PHYSPAGES_BARRIER BC_MAXVALUE
+#define BC_PHYSPAGES_LIMIT BC_MAXVALUE
+
+/*
+ * page_beancounter is a tie between page and beancounter page is
+ * charged to. it is used to reclaim pages faster by walking bc's
+ * page list, not zones' ones
+ *
+ * this tie can also be used to implement fractions accounting mechanism
+ * as it is done in OpenVZ kernels
+ */
+struct page_beancounter {
+ struct page *page;
+ struct beancounter *bc;
+ struct list_head list;
+};
+
+/*
+ * API calls
+ */
+
+/*
+ * bc_rsspage_prepare allocates a tie and charges page to vma's beancounter
+ * this must be called in non-atomic context to give a chance for pages
+ * reclaiming. otherwise hitting limits will cause -ENOMEM returned.
+ */
+int bc_rsspage_prepare(struct page *page, struct vm_area_struct *vma,
+  struct page_beancounter **ppb)
+{
+ struct beancounter *bc;
+ struct page_beancounter *pb;
+
+ pb = kmalloc(sizeof(struct page_beancounter), GFP_KERNEL);
+ if (pb == NULL)
+  goto out_nomem;
+
+ bc = vma->vma_bc;
+ if (bc_charge(bc, BC_PHYSPAGES, 1, BC_LIMIT))
+  goto out_charge;
+
+ pb->page = page;
+ pb->bc = bc;
+ *ppb = pb;
```

```
+ return 0;
+
+out_charge:
+ kfree(pb);
+out_nomem:
+ return -ENOMEM;
+}
+
+/*
+ * bc_rsspage_release is a rollback call for bc_rsspage_prepare
+ */
+void bc_rsspage_release(struct page_beancounter *pb)
+{
+ bc_uncharge(pb->bc, BC_PHYSPAGES, 1);
+ kfree(pb);
+}
+
+/*
+ * bc_rsspage_charge actually ties page and beancounter together
+ * this is done in not-failing path to be sure the page IS charged
+ */
+void bc_rsspage_charge(struct page_beancounter *pb)
+{
+ struct page *pg;
+ struct beancounter *bc;
+
+ pg = pb->page;
+ bc = bc_get(pb->bc);
+
+ spin_lock(&bc->bc_page_lock);
+ list_add(&pb->list, &bc->bc_page_list);
+ spin_unlock(&bc->bc_page_lock);
+
+ page_pb(pg) = pb;
+}
+
+/*
+ * bc_rsspage_uncharge is called when pages is get completely unapped
+ * from all address spaces
+ */
+void bc_rsspage_uncharge(struct page_beancounter *pb)
+{
+ struct page *page;
+ struct beancounter *bc;
+
+ if (pb == NULL)
+  return;
+
```

```
+ page = pb->page;
+ bc = pb->bc;
+
+ cmpxchg(&page_pb(page), pb, NULL);
+
+ spin_lock(&bc->bc_page_lock);
+ list_del(&pb->list);
+ spin_unlock(&bc->bc_page_lock);
+
+ bc_uncharge(bc, BC_PHYSPAGES, 1);
+ kfree(pb);
+
+ bc_put(bc);
+}
+
+/*
+ * Page reclamation helper
+ *
+ * this function resembles isolate_lru_pages() but is scans through
+ * bc's page list, not zone's active/inactive ones.
+ */
+
+unsigned long bc_isolate_pages(unsigned long nr_to_scan, struct beancounter *bc,
+  struct list_head *dst, int active, unsigned long *scanned)
+{
+ unsigned long nr_taken = 0;
+ struct page *page;
+ struct page_beancounter *pb;
+ unsigned long scan;
+ struct list_head *src;
+ LIST_HEAD(pb_list);
+ struct zone *z;
+
+ spin_lock(&bc->bc_page_lock);
+ src = &bc->bc_page_list;
+ for (scan = 0; scan < nr_to_scan && !list_empty(src); scan++) {
+  struct list_head *target;
+  pb = list_entry(src->prev, struct page_beancounter, list);
+  page = pb->page;
+  z = page_zone(page);
+
+  list_move(&pb->list, &pb_list);
+
+  spin_lock_irq(&z->lru_lock);
+  if (PageLRU(page)) {
+   if ((active && PageActive(page)) ||
+     (!active && !PageActive(page))) {
+    if (likely(get_page_unless_zero(page))) {
```

```
+     ClearPageLRU(page);
+     target = dst;
+     nr_taken++;
+     list_move(&page->lru, dst);
+    }
+   }
+  }
+  spin_unlock_irq(&z->lru_lock);
+ }
+
+ list_splice(&pb_list, src);
+ spin_unlock(&bc->bc_page_lock);
+
+ *scanned = scan;
+ return nr_taken;
+}
+
+unsigned long bc_nr_physpages(struct beancounter *bc)
+{
+ return bc->bc_parms[BC_PHYSPAGES].held;
+}
+
+/*
+ * Generic resource info
+ */
+
+static int bc_phys_init(struct beancounter *bc, int res)
+{
+ spin_lock_init(&bc->bc_page_lock);
+ INIT_LIST_HEAD(&bc->bc_page_list);
+
+ bc_init_resource(&bc->bc_parms[BC_PHYSPAGES],
+   BC_PHYSPAGES_BARRIER, BC_PHYSPAGES_LIMIT);
+ return 0;
+}
+
+static void bc_phys_barrier_hit(struct beancounter *bc)
+{
+ /*
+  * May wake up kswapd here to start asynchronous reclaiming of pages
+  */
+}
+
+static int bc_phys_limit_hit(struct beancounter *bc, unsigned long val,
+  unsigned long flags)
+{
+ int did_some_progress = 0;
+ struct bc_resource_parm *parm;
```

```
+
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ might_sleep();
+
+ parm = &bc->bc_parms[BC_PHYSPAGES];
+ while (1) {
+  did_some_progress = bc_try_to_free_pages(bc);
+
+  spin_lock_irq(&bc->bc_lock);
+  if (parm->held + val <= parm->limit) {
+   parm->held += val;
+   bc_adjust_maxheld(parm);
+   return 0;
+  }
+
+  if (!did_some_progress) {
+   parm->failcnt++;
+   return -ENOMEM;
+  }
+  spin_unlock_irq(&bc->bc_lock);
+ }
+}
+
+static int bc_phys_change(struct beancounter *bc,
+  unsigned long barrier, unsigned long limit)
+{
+ int did_some_progress;
+ struct bc_resource_parm *parm;
+
+ parm = &bc->bc_parms[BC_PHYSPAGES];
+ if (limit >= parm->held)
+  return 0;
+
+ while (1) {
+  spin_unlock_irq(&bc->bc_lock);
+
+  did_some_progress = bc_try_to_free_pages(bc);
+
+  spin_lock_irq(&bc->bc_lock);
+  if (parm->held < limit)
+   return 0;
+  if (!did_some_progress)
+   return -ENOMEM;
+ }
+}
+
+struct bc_resource bc_phys_resource = {
+ .bcr_name  = "physpages",
```

```
+ .bcr_init  = bc_phys_init,
+ .bcr_change  = bc_phys_change,
+ .bcr_barrier_hit = bc_phys_barrier_hit,
+ .bcr_limit_hit  = bc_phys_limit_hit,
+};
+
+static int __init bc_phys_init_resource(void)
+{
+ bc_register_resource(BC_PHYSPAGES, &bc_phys_resource);
+ return 0;
+}
+
+__initcall(bc_phys_init_resource);
```