Subject: [PATCH 8/13] BC: privvmpages accounting (core)
Posted by dev on Thu, 09 Nov 2006 16:56:01 GMT
View Forum Message <> Reply to Message

This patch instroduces new resource - BC_PRIVVMPAGES.
It is an upper estimation of currently used physical memory.

There are different approaches to user pages control:
a) account all the mappings on mmap/brk and reject as
   soon as the sum of VMA's lengths reaches the barrier.

   This approach is very bad as applications always map
   more than they really use, very often MUCH more.

b) account only the really used memory and reject as
   soon as RSS reaches the limit.

   This approach is not good either as user space pages are
   allocated in page fault handler and the only way to reject
   allocation is to kill the task.

   Comparing to previous scenarion this is much worse as
   application won't even be able to terminate gracefully.

c) account a part of memory on mmap/brk and reject there,
   and account the rest of the memory in page fault handlers
   without any rejects.
   This type of accounting is used in UBC.

d) account physical memory and behave like a standalone
   kernel - reclaim user memory when run out of it.

   This type of memory control is to be introduced later
   as an addition to c). UBC provides all the needed
   statistics for this (physical memory, swap pages etc.)

Privvmpages accounting is described in details in
http://wiki.openvz.org/User_pages_accounting

A note about sys_mprotect: as it can change mapping state from
bc_vm_private to !bc_vm_private and vice-versa appropriate amount of
pages is (un)charged in mprotect_fixup.

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

```
 include/bc/vmpages.h    |   90 +++++++++++++++++++++++++++++++
 include/linux/mm.h      |    3 +
 include/linux/sched.h   |    3 +
 kernel/bc/beancounter.c |    1
 kernel/bc/vmpages.c     |  138 ++++++++++++++++++++++++++++++++++++++++++++++++
 5 files changed, 235 insertions(+)

--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./include/bc/vmpages.h 2006-11-03 17:49:13.000000000 +0300
@@ -0,0 +1,90 @@
+/*
+ * include/bc/vmpages.h
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
+#ifndef __BC_VMPAGES_H_
+#define __BC_VMPAGES_H_
+
+#include <bc/beancounter.h>
+
+struct vm_area_struct;
+struct mm_struct;
+struct file;
+
+#define BC_NOCHARGE 0
+#define BC_UNCHARGE 1
+#define BC_CHARGE 2
+
+#ifdef CONFIG_BEANCOUNTERS
+#define __vma_set_bc(vma, bc) do { (vma)->vma_bc = bc_get(bc); } while (0)
+#define vma_set_bc(vma) __vma_set_bc(vma, (vma)->vm_mm->mm_bc)
+#define vma_copy_bc(vma) __vma_set_bc(vma, (vma)->vma_bc)
+#define vma_release_bc(vma) do { bc_put((vma)->vma_bc); } while (0)
+
+#define mm_init_beancounter(mm) do {  \
+  struct beancounter *bc;  \
+  bc = get_exec_bc();  \
+  (mm)->mm_bc = bc_get(bc); \
+ } while (0)
+#define mm_free_beancounter(mm) do { bc_put(mm->mm_bc); } while (0)
+
+int __must_check bc_need_memory_recharge(struct vm_area_struct *vma,
+  struct file *new_file, unsigned long new_flags);
+
+int __must_check __bc_memory_charge(struct mm_struct *mm, unsigned long len,
+  int severity);
```

```
+int __must_check bc_memory_charge(struct mm_struct *mm, unsigned long len,
+  struct file *file, unsigned long flags, int severity);
+int __must_check bc_vma_charge(struct vm_area_struct *vma);
+
+void __bc_memory_uncharge(struct mm_struct *mm, unsigned long len);
+void bc_memory_uncharge(struct mm_struct *mm, unsigned long len,
+  struct file *file, unsigned long flags);
+void bc_vma_uncharge(struct vm_area_struct *vma);
+
+#define bc_equal(bc1, bc2) (bc1 == bc2)
+#else
+static inline
+int __must_check bc_need_memory_recharge(struct vm_area_struct *vma,
+  struct file *new_file, unsigned long new_flags)
+{
+ return BC_NOCHARGE;
+}
+static inline int __must_check __bc_memory_charge(struct mm_struct *mm,
+  unsigned long len, int severity)
+{
+ return 0;
+}
+static inline int __must_check bc_memory_charge(struct mm_struct *mm,
+  unsigned long len, struct file *file, unsigned long flags,
+  int severity)
+{
+ return 0;
+}
+static inline int __must_check bc_vma_charge(struct vm_area_struct *vma)
+{
+ return 0;
+}
+static inline void __bc_memory_uncharge(struct mm_struct *mm, unsigned long len)
+{
+}
+static inline void bc_memory_uncharge(struct mm_struct *mm, unsigned long len,
+  struct file *file, unsigned long flags)
+{
+}
+static inline void bc_vma_uncharge(struct vm_area_struct *vma)
+{
+}
+
+#define mm_init_beancounter(mm) do { } while (0)
+#define mm_free_beancounter(mm) do { } while (0)
+#define __vma_set_bc(vma, bc) do { } while (0)
+#define vma_set_bc(vma)  do { } while (0)
+#define vma_copy_bc(vma) do { } while (0)
```

```
+#define vma_release_bc(vma) do { } while (0)
+#define bc_equal(bc1, bc2) 1
+#endif
+#endif
--- ./include/linux/mm.h.bcvmpcore 2006-11-03 17:48:37.000000000 +0300
+++ ./include/linux/mm.h 2006-11-03 17:49:13.000000000 +0300
@@ -112,6 +112,9 @@ struct vm_area_struct {
 #ifdef CONFIG_NUMA
  struct mempolicy *vm_policy; /* NUMA policy for the VMA */
 #endif
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *vma_bc;
+#endif
 };

 /*
--- ./include/linux/sched.h.bcvmpcore 2006-11-03 17:47:38.000000000 +0300
+++ ./include/linux/sched.h 2006-11-03 17:49:13.000000000 +0300
@@ -374,6 +374,9 @@ struct mm_struct {
  /* aio bits */
  rwlock_t  ioctx_list_lock;
  struct kioctx  *ioctx_list;
+#ifdef CONFIG_BEANCOUNTERS
+ struct beancounter *mm_bc;
+#endif
 };

 struct sighand_struct {
--- ./kernel/bc/beancounter.c.bcvmpcore 2006-11-03 17:47:38.000000000 +0300
+++ ./kernel/bc/beancounter.c 2006-11-03 17:49:36.000000000 +0300
@@ -237,6 +237,7 @@ void __init bc_init_early(void)
  hlist_add_head(&init_bc.bc_hash, &bc_hash[hash_long(0, BC_HASH_BITS)]);

  current->exec_bc = bc_get(&init_bc);
+ init_mm.mm_bc = bc_get(&init_bc);
 }

 int __init bc_init_late(void)
--- /dev/null 2006-07-18 14:52:43.075228448 +0400
+++ ./kernel/bc/vmpages.c 2006-11-03 17:49:13.000000000 +0300
@@ -0,0 +1,138 @@
+/*
+ * kernel/bc/vmpages.c
+ *
+ * Copyright (C) 2006 OpenVZ SWsoft Inc
+ *
+ */
+
```

```
+#include <linux/mm.h>
+
+#include <bc/beancounter.h>
+#include <bc/vmpages.h>
+
+#define BC_PRIVVMPAGES_BARRIER BC_MAXVALUE
+#define BC_PRIVVMPAGES_LIMIT BC_MAXVALUE
+
+/*
+ * Core routines
+ */
+
+/*
+ * bc_vma_private checks whether VMA (file, flags) is private
+ * from BC point of view. private VMAs are charged when they are mapped
+ * thus prventing system from resource exhausting when pages from these VMAs
+ * are touched.
+ */
+static inline int bc_vma_private(struct file *file, unsigned long flags)
+{
+ return (flags & VM_LOCKED) ||
+  ((flags & VM_WRITE) && (file == NULL || !(flags & VM_SHARED)));
+}
+
+/*
+ * Accounting is performed in pages (not in Kbytes)
+ */
+static inline int do_memory_charge(struct beancounter *bc,
+  unsigned long len, int severity)
+{
+ return bc_charge(bc, BC_PRIVVMPAGES, len >> PAGE_SHIFT, severity);
+}
+
+static inline void do_memory_uncharge(struct beancounter *bc, unsigned long len)
+{
+ bc_uncharge(bc, BC_PRIVVMPAGES, len >> PAGE_SHIFT);
+}
+
+/*
+ * API calls
+ */
+
+int __bc_memory_charge(struct mm_struct *mm, unsigned long len, int severity)
+{
+ return do_memory_charge(mm->mm_bc, len, severity);
+}
+
+int bc_memory_charge(struct mm_struct *mm, unsigned long len,
```

```
+  struct file *file, unsigned long flags, int severity)
+{
+ int ret;
+
+ ret = 0;
+ if (bc_vma_private(file, flags))
+  ret = do_memory_charge(mm->mm_bc, len, severity);
+ return ret;
+}
+
+int bc_vma_charge(struct vm_area_struct *vma)
+{
+ int ret;
+
+ ret = (bc_vma_private(vma->vm_file, vma->vm_flags) ?
+   do_memory_charge(vma->vm_mm->mm_bc,
+    vma->vm_end - vma->vm_start, BC_BARRIER) : 0);
+ if (ret == 0)
+  vma_set_bc(vma);
+ return ret;
+}
+
+void __bc_memory_uncharge(struct mm_struct *mm, unsigned long len)
+{
+ do_memory_uncharge(mm->mm_bc, len);
+}
+
+void bc_memory_uncharge(struct mm_struct *mm, unsigned long len,
+  struct file *file, unsigned long flags)
+{
+ if (bc_vma_private(file, flags))
+  do_memory_uncharge(mm->mm_bc, len);
+}
+
+void bc_vma_uncharge(struct vm_area_struct *vma)
+{
+ if (bc_vma_private(vma->vm_file, vma->vm_flags))
+  do_memory_uncharge(vma->vma_bc, vma->vm_end - vma->vm_start);
+ vma_release_bc(vma);
+}
+
+
+int bc_need_memory_recharge(struct vm_area_struct *vma, struct file *new_file,
+  unsigned long new_flags)
+{
+ if (bc_vma_private(vma->vm_file, vma->vm_flags)) {
+  if (bc_vma_private(new_file, new_flags))
+   return BC_NOCHARGE;
```

```
+
+  /* private -> non-private */
+  return BC_UNCHARGE;
+ } else {
+  if (!bc_vma_private(new_file, new_flags))
+   return BC_NOCHARGE;
+
+  /* non-private -> private */
+  return BC_CHARGE;
+ }
+}
+
+/*
+ * Generic resource info
+ */
+
+static int bc_privvm_init(struct beancounter *bc, int res)
+{
+ bc_init_resource(&bc->bc_parms[BC_PRIVVMPAGES],
+   BC_PRIVVMPAGES_BARRIER, BC_PRIVVMPAGES_LIMIT);
+ return 0;
+}
+
+struct bc_resource bc_privvm_resource = {
+ .bcr_name  = "privvmpages",
+ .bcr_init  = bc_privvm_init,
+};
+
+static int __init bc_privvm_init_resource(void)
+{
+ bc_register_resource(BC_PRIVVMPAGES, &bc_privvm_resource);
+ return 0;
+}
+
+__initcall(bc_privvm_init_resource);
```