Subject: [PATCH 10/13] BC: privvm pages
Posted by dev on Tue, 05 Sep 2006 15:26:58 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/beancounter.h |    3 +-
 include/bc/vmpages.h     |   44 ++++++++++++++++++++++++++++++++++++++++++++
 kernel/bc/beancounter.c  |    2 +
 kernel/bc/vmpages.c      |   53 ++++++++++++++++++++++++++++++++++++++++++++++++---
 kernel/fork.c            |    9 +++++++
 mm/mprotect.c            |   17 ++++++++++++++-
 mm/shmem.c               |    7 ++++++
 7 files changed, 129 insertions(+), 6 deletions(-)

--- ./include/bc/beancounter.h.bcprivvm 2006-09-05 12:59:27.000000000 +0400
+++ ./include/bc/beancounter.h 2006-09-05 13:17:50.000000000 +0400
@@ -14,8 +14,9 @@

 #define BC_KMEMSIZE 0
 #define BC_LOCKEDPAGES 1
+#define BC_PRIVVMPAGES 2

-#define BC_RESOURCES 2
+#define BC_RESOURCES 3

 struct bc_resource_parm {
  unsigned long barrier; /* A barrier over which resource allocations
--- ./include/bc/vmpages.h.bcprivvm 2006-09-05 13:04:03.000000000 +0400
+++ ./include/bc/vmpages.h 2006-09-05 13:38:07.000000000 +0400
@@ -8,6 +8,8 @@
 #ifndef __BC_VMPAGES_H_
 #define __BC_VMPAGES_H_

+#include <linux/mm.h>
+
 #include <bc/beancounter.h>
 #include <bc/task.h>

@@ -15,12 +17,37 @@ struct mm_struct;
 struct file;
 struct shmem_inode_info;

+/*
+ * sys_mprotect() can change mapping state form private to
+ * shared and vice-versa. Thus rescharging is needed, but
+ * with the following rules:
+ * 1. No state change   : nothing to be done at all;
+ * 2. shared -> private : need to charge before operation starts
+ *                        and roll back on error path;
+ * 3. private -> shared : need to uncharge after successfull state
+ *                        change. Uncharging first and charging back
+ *                        on error path isn't good as charge will have
+ *                        to be BC_FORCE and thus can potentially create
```

```
+ *                      an overcharged privvmpages.
+ */
+#define BC_NOCHARGE 0
+#define BC_UNCHARGE 1 /* private -> shared */
+#define BC_CHARGE 2 /* shared -> private */
+
+#define BC_VM_PRIVATE(flags, file) ( ((flags) & VM_WRITE) ? \
+   ( (file) == NULL || !((flags) & VM_SHARED) ) : 0 )
+
 #ifdef CONFIG_BEANCOUNTERS
 int __must_check bc_memory_charge(struct mm_struct *mm, unsigned long size,
   unsigned long vm_flags, struct file *vm_file, int strict);
 void bc_memory_uncharge(struct mm_struct *mm, unsigned long size,
   unsigned long vm_flags, struct file *vm_file);

+int __must_check bc_privvm_recharge(unsigned long old_flags,
+  unsigned long new_flags, struct file *vm_file);
+int __must_check bc_privvm_charge(struct mm_struct *mm, unsigned long size);
+void bc_privvm_uncharge(struct mm_struct *mm, unsigned long size);
+
 int __must_check bc_locked_charge(struct mm_struct *mm, unsigned long size);
 void bc_locked_uncharge(struct mm_struct *mm, unsigned long size);

@@ -64,6 +91,23 @@ static inline void bc_memory_uncharge(st
 {
 }

+static inline int __must_check bc_privvm_recharge(unsigned long old_flags,
+  unsigned long new_flags, struct file *vm_file)
+{
+ return BC_NOCHARGE;
+}
+
+static inline int __must_check bc_privvm_charge(struct mm_struct *mm,
+  unsigned long size)
+{
+ return 0;
+}
+
+static inline void bc_privvm_uncharge(struct mm_struct *mm,
+  unsigned long size)
+{
+}
+
 static inline int __must_check bc_locked_charge(struct mm_struct *mm,
   unsigned long size)
 {
--- ./kernel/bc/beancounter.c.bcprivvm 2006-09-05 12:59:45.000000000 +0400
```

```
+++ ./kernel/bc/beancounter.c 2006-09-05 13:17:50.000000000 +0400
@@ -22,6 +22,7 @@ struct beancounter init_bc;
 const char *bc_rnames[] = {
  "kmemsize", /* 0 */
  "lockedpages",
+ "privvmpages",
 };

 #define BC_HASH_BITS  8
@@ -234,6 +235,7 @@ static void init_beancounter_syslimits(s

  bc->bc_parms[BC_KMEMSIZE].limit = 32 * 1024 * 1024;
  bc->bc_parms[BC_LOCKEDPAGES].limit = 8;
+ bc->bc_parms[BC_PRIVVMPAGES].limit = BC_MAXVALUE;

  for (k = 0; k < BC_RESOURCES; k++)
   bc->bc_parms[k].barrier = bc->bc_parms[k].limit;
--- ./kernel/bc/vmpages.c.bcprivvm 2006-09-05 12:59:27.000000000 +0400
+++ ./kernel/bc/vmpages.c 2006-09-05 13:28:16.000000000 +0400
@@ -18,26 +18,73 @@ int bc_memory_charge(struct mm_struct *m
   unsigned long vm_flags, struct file *vm_file, int strict)
 {
  struct beancounter *bc;
+ unsigned long flags;

  bc = mm->mm_bc;
  size >>= PAGE_SHIFT;

+ spin_lock_irqsave(&bc->bc_lock, flags);
  if (vm_flags & VM_LOCKED)
-  if (bc_charge(bc, BC_LOCKEDPAGES, size, strict))
-   return -ENOMEM;
+  if (bc_charge_locked(bc, BC_LOCKEDPAGES, size, strict))
+   goto err_locked;
+ if (BC_VM_PRIVATE(vm_flags, vm_file))
+  if (bc_charge_locked(bc, BC_PRIVVMPAGES, size, strict))
+   goto err_privvm;
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
  return 0;
+
+err_privvm:
+ bc_uncharge_locked(bc, BC_LOCKEDPAGES, size);
+err_locked:
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+ return -ENOMEM;
 }

 void bc_memory_uncharge(struct mm_struct *mm, unsigned long size,
```

```
   unsigned long vm_flags, struct file *vm_file)
 {
  struct beancounter *bc;
+ unsigned long flags;

  bc = mm->mm_bc;
  size >>= PAGE_SHIFT;

+ spin_lock_irqsave(&bc->bc_lock, flags);
  if (vm_flags & VM_LOCKED)
-  bc_uncharge(bc, BC_LOCKEDPAGES, size);
+  bc_uncharge_locked(bc, BC_LOCKEDPAGES, size);
+ if (BC_VM_PRIVATE(vm_flags, vm_file))
+  bc_uncharge_locked(bc, BC_PRIVVMPAGES, size);
+ spin_unlock_irqrestore(&bc->bc_lock, flags);
+}
+
+int bc_privvm_recharge(unsigned long vm_flags_old, unsigned long vm_flags_new,
+  struct file *vm_file)
+{
+ int priv_old, priv_new;
+
+ priv_old = (BC_VM_PRIVATE(vm_flags_old, vm_file) ? 1 : 0);
+ priv_new = (BC_VM_PRIVATE(vm_flags_new, vm_file) ? 1 : 0);
+
+ if (priv_old == priv_new)
+  return BC_NOCHARGE;
+
+ return priv_new ? BC_CHARGE : BC_UNCHARGE;
+}
+
+int bc_privvm_charge(struct mm_struct *mm, unsigned long size)
+{
+ struct beancounter *bc;
+
+ bc = mm->mm_bc;
+ bc_charge(bc, BC_PRIVVMPAGES, size >> PAGE_SHIFT);
+}
+
+void bc_privvm_uncharge(struct mm_struct *mm, unsigned long size)
+{
+ struct beancounter *bc;
+
+ bc = mm->mm_bc;
+ bc_uncharge(bc, BC_PRIVVMPAGES, size >> PAGE_SHIFT);
 }

 static inline int locked_charge(struct beancounter *bc,
```

```
--- ./kernel/fork.c.bcprivvm 2006-09-05 13:17:15.000000000 +0400
+++ ./kernel/fork.c 2006-09-05 13:23:27.000000000 +0400
@@ -236,9 +236,13 @@ static inline int dup_mmap(struct mm_str
     goto fail_nomem;
    charge = len;
   }
+  if (bc_memory_charge(mm, mpnt->vm_end - mpnt->vm_start,
+     mpnt->vm_flags & ~VM_LOCKED,
+     mpnt->vm_file, BC_LIMIT) < 0)
+   goto fail_nomem;
   tmp = kmem_cache_alloc(vm_area_cachep, SLAB_KERNEL);
   if (!tmp)
-   goto fail_nomem;
+   goto fail_alloc;
   *tmp = *mpnt;
   pol = mpol_copy(vma_policy(mpnt));
   retval = PTR_ERR(pol);
@@ -292,6 +296,9 @@ out:
  return retval;
 fail_nomem_policy:
  kmem_cache_free(vm_area_cachep, tmp);
+fail_alloc:
+ bc_memory_uncharge(mm, mpnt->vm_end - mpnt->vm_start,
+   mpnt->vm_flags & ~VM_LOCKED, mpnt->vm_file);
 fail_nomem:
  retval = -ENOMEM;
  vm_unacct_memory(charge);
--- ./mm/mprotect.c.bcprivvm 2006-09-05 12:53:59.000000000 +0400
+++ ./mm/mprotect.c 2006-09-05 13:27:40.000000000 +0400
@@ -21,6 +21,7 @@
 #include <linux/syscalls.h>
 #include <linux/swap.h>
 #include <linux/swapops.h>
+#include <bc/vmpages.h>
 #include <asm/uaccess.h>
 #include <asm/pgtable.h>
 #include <asm/cacheflush.h>
@@ -139,12 +140,19 @@ mprotect_fixup(struct vm_area_struct *vm
  pgoff_t pgoff;
  int error;
  int dirty_accountable = 0;
+ int recharge;

  if (newflags == oldflags) {
   *pprev = vma;
   return 0;
  }
```

```
+ recharge = bc_privvm_recharge(oldflags, newflags, vma->vm_file);
+ if (recharge == BC_CHARGE) {
+  if (bc_privvm_charge(mm, end - start))
+   return -ENOMEM;
+ }
+
  /*
   * If we make a private mapping writable we increase our commit;
   * but (without finer accounting) cannot reduce our commit if we
@@ -157,8 +165,9 @@ mprotect_fixup(struct vm_area_struct *vm
  if (newflags & VM_WRITE) {
   if (!(oldflags & (VM_ACCOUNT|VM_WRITE|VM_SHARED))) {
    charged = nrpages;
+   error = -ENOMEM;
    if (security_vm_enough_memory(charged))
-    return -ENOMEM;
+    goto fail_acct;
    newflags |= VM_ACCOUNT;
   }
  }
@@ -205,12 +213,18 @@ success:
   hugetlb_change_protection(vma, start, end, vma->vm_page_prot);
  else
   change_protection(vma, start, end, vma->vm_page_prot, dirty_accountable);
+
+ if (recharge == BC_UNCHARGE)
+  bc_privvm_uncharge(mm, end - start);
  vm_stat_account(mm, oldflags, vma->vm_file, -nrpages);
  vm_stat_account(mm, newflags, vma->vm_file, nrpages);
  return 0;

 fail:
  vm_unacct_memory(charged);
+fail_acct:
+ if (recharge == BC_CHARGE)
+  bc_privvm_uncharge(mm, end - start);
  return error;
 }

--- ./mm/shmem.c.bcprivvm 2006-09-05 13:06:37.000000000 +0400
+++ ./mm/shmem.c 2006-09-05 13:39:26.000000000 +0400
@@ -2363,6 +2363,13 @@ int shmem_zero_setup(struct vm_area_stru

  if (vma->vm_file)
   fput(vma->vm_file);
+ else if (vma->vm_flags & VM_WRITE)
+  /*
+   * this means that mapping was considered to be private
```

```
+   * in do_mmap_pgoff, but now it becomes non-private, as
+   * file is attached to the vma.
+   */
+  bc_privvm_uncharge(vma->vm_mm, size);
  vma->vm_file = file;
  vma->vm_ops = &shmem_vm_ops;
  return 0;
```