Subject: [RFC PATCH 4/5] use the memcg VM overcommit prototypes
Posted by Andrea Righi on Mon, 09 Jun 2008 23:33:02 GMT
View Forum Message <> Reply to Message

Apply the new memory controller VM prototypes to the opportune kernel routines.

Signed-off-by: Andrea Righi <righi.andrea@gmail.com>
---
 ipc/shm.c | 6 ++++--
 kernel/fork.c | 2 +-
 mm/mmap.c | 10 ++++++++----
 mm/mprotect.c | 2 +-
 mm/mremap.c | 4 ++--
 mm/shmem.c | 49 ++---------
 mm/swapfile.c | 4 ++--
 7 files changed, 57 insertions(+), 20 deletions(-)

```
diff --git a/ipc/shm.c b/ipc/shm.c
index 554429a..420bfa9 100644
--- a/ipc/shm.c
+++ b/ipc/shm.c
@@ -384,12 +384,14 @@ static int newseg(struct ipc_namespace *ns, struct ipc_params
*params)
   shp->mlock_user = current->user;
  } else {
   int acctflag = VM_ACCOUNT;
+  struct vm_acct_values v;
   /*
    * Do not allow no accounting for OVERCOMMIT_NEVER, even
-    * if it's asked for.
+   * if it's asked for.
    */
+  vm_acct_get_config(current->mm, &v);
   if  ((shmflg & SHM_NORESERVE) &&
-    sysctl_overcommit_memory != OVERCOMMIT_NEVER)
+    v.overcommit_memory != OVERCOMMIT_NEVER)
    acctflag = 0;
   file = shmem_file_setup(name, size, acctflag);
  }
diff --git a/kernel/fork.c b/kernel/fork.c
index 19908b2..eaffa56 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -336,7 +336,7 @@ fail_nomem_policy:
  kmem_cache_free(vm_area_cachep, tmp);
 fail_nomem:
  retval = -ENOMEM;
- vm_unacct_memory(charge);
```

```
+ vm_unacct_memory(mm, charge);
  goto out;
 }

diff --git a/mm/mmap.c b/mm/mmap.c
index 256599e..ab40277 100644
--- a/mm/mmap.c
+++ b/mm/mmap.c
@@ -1028,6 +1028,7 @@ unsigned long mmap_region(struct file *file, unsigned long addr,
  int error;
  struct rb_node **rb_link, *rb_parent;
  unsigned long charged = 0;
+ struct vm_acct_values v;
  struct inode *inode =  file ? file->f_path.dentry->d_inode : NULL;

  /* Clear old maps */
@@ -1044,6 +1045,7 @@ munmap_back:
  if (!may_expand_vm(mm, len >> PAGE_SHIFT))
   return -ENOMEM;

+ vm_acct_get_config(mm, &v);
  if (accountable && (!(flags & MAP_NORESERVE) ||
       sysctl_overcommit_memory == OVERCOMMIT_NEVER)) {
   if (vm_flags & VM_SHARED) {
@@ -1170,7 +1172,7 @@ free_vma:
  kmem_cache_free(vm_area_cachep, vma);
 unacct_error:
  if (charged)
-  vm_unacct_memory(charged);
+  vm_unacct_memory(mm, charged);
  return error;
 }

@@ -1698,7 +1700,7 @@ static void unmap_region(struct mm_struct *mm,
  tlb = tlb_gather_mmu(mm, 0);
  update_hiwater_rss(mm);
  unmap_vmas(&tlb, vma, start, end, &nr_accounted, NULL);
- vm_unacct_memory(nr_accounted);
+ vm_unacct_memory(mm, nr_accounted);
  free_pgtables(&tlb, vma, prev? prev->vm_end: FIRST_USER_ADDRESS,
      next? next->vm_start: 0);
  tlb_finish_mmu(tlb, start, end);
@@ -1959,7 +1961,7 @@ unsigned long do_brk(unsigned long addr, unsigned long len)
   */
  vma = kmem_cache_zalloc(vm_area_cachep, GFP_KERNEL);
  if (!vma) {
-  vm_unacct_memory(len >> PAGE_SHIFT);
+  vm_unacct_memory(mm, len >> PAGE_SHIFT);
```

```
   return -ENOMEM;
  }

@@ -1998,7 +2000,7 @@ void exit_mmap(struct mm_struct *mm)
  /* Don't update_hiwater_rss(mm) here, do_exit already did */
  /* Use -1 here to ensure all VMAs in the mm are unmapped */
  end = unmap_vmas(&tlb, vma, 0, -1, &nr_accounted, NULL);
- vm_unacct_memory(nr_accounted);
+ vm_unacct_memory(mm, nr_accounted);
  free_pgtables(&tlb, vma, FIRST_USER_ADDRESS, 0);
  tlb_finish_mmu(tlb, 0, end);

diff --git a/mm/mprotect.c b/mm/mprotect.c
index a5bf31c..2b1ad3b 100644
--- a/mm/mprotect.c
+++ b/mm/mprotect.c
@@ -216,7 +216,7 @@ success:
  return 0;

 fail:
- vm_unacct_memory(charged);
+ vm_unacct_memory(mm, charged);
  return error;
 }

diff --git a/mm/mremap.c b/mm/mremap.c
index 08e3c7f..9dc5921 100644
--- a/mm/mremap.c
+++ b/mm/mremap.c
@@ -217,7 +217,7 @@ static unsigned long move_vma(struct vm_area_struct *vma,

  if (do_munmap(mm, old_addr, old_len) < 0) {
   /* OOM: unable to split vma, just get accounts right */
-  vm_unacct_memory(excess >> PAGE_SHIFT);
+  vm_unacct_memory(mm, excess >> PAGE_SHIFT);
   excess = 0;
  }
  mm->hiwater_vm = hiwater_vm;
@@ -407,7 +407,7 @@ unsigned long do_mremap(unsigned long addr,
  }
 out:
  if (ret & ~PAGE_MASK)
-  vm_unacct_memory(charged);
+  vm_unacct_memory(mm, charged);
 out_nc:
  return ret;
 }
diff --git a/mm/shmem.c b/mm/shmem.c
```

```
index e2a6ae1..2cd56be 100644
--- a/mm/shmem.c
+++ b/mm/shmem.c
@@ -163,14 +163,30 @@ static inline struct shmem_sb_info *SHMEM_SB(struct super_block
*sb)
  */
 static inline int shmem_acct_size(unsigned long flags, loff_t size)
 {
- return (flags & VM_ACCOUNT)?
-  security_vm_enough_memory(VM_ACCT(size)): 0;
+ if (!(flags & VM_ACCOUNT))
+  return 0;
+ /*
+  * TODO: find a way to correctly account shared memory also for the
+  * cgroup memory controller, as well as non-shared memory.  For now
+  * simply undo the accounting in the cgroup's committed space and
+  * report the size only in the global committed memory.
+  */
+ mem_cgroup_vm_acct_memory(current->mm, -VM_ACCT(size));
+ return security_vm_enough_memory(VM_ACCT(size));
 }

 static inline void shmem_unacct_size(unsigned long flags, loff_t size)
 {
- if (flags & VM_ACCOUNT)
-  vm_unacct_memory(VM_ACCT(size));
+ /*
+  * TODO: find a way to correctly account shared memory also for the
+  * cgroup memory controller, as well as non-shared memory.  For now
+  * simply undo the accounting in the cgroup's committed space and
+  * report the size only in the global committed memory.
+  */
+ if (!(flags & VM_ACCOUNT))
+  return;
+ mem_cgroup_vm_acct_memory(current->mm, VM_ACCT(size));
+ vm_unacct_memory(current->mm, VM_ACCT(size));
 }

 /*
@@ -181,14 +197,31 @@ static inline void shmem_unacct_size(unsigned long flags, loff_t size)
  */
 static inline int shmem_acct_block(unsigned long flags)
 {
- return (flags & VM_ACCOUNT)?
-  0: security_vm_enough_memory(VM_ACCT(PAGE_CACHE_SIZE));
+ /*
+  * TODO: find a way to correctly account shared memory also for the
+  * cgroup memory controller, as well as non-shared memory.  For now
```

```
+  * simply undo the accounting in the cgroup's committed space and
+  * report the size only in the global committed memory.
+  */
+ if (flags & VM_ACCOUNT)
+  return 0;
+ mem_cgroup_vm_acct_memory(current->mm, -VM_ACCT(PAGE_CACHE_SIZE));
+ return security_vm_enough_memory(VM_ACCT(PAGE_CACHE_SIZE));
 }

 static inline void shmem_unacct_blocks(unsigned long flags, long pages)
 {
- if (!(flags & VM_ACCOUNT))
-  vm_unacct_memory(pages * VM_ACCT(PAGE_CACHE_SIZE));
+ /*
+  * TODO: find a way to correctly account shared memory also for the
+  * cgroup memory controller, as well as non-shared memory.  For now
+  * simply undo the accounting in the cgroup's committed space and
+  * report the size only in the global committed memory.
+  */
+ if (flags & VM_ACCOUNT)
+  return;
+ mem_cgroup_vm_acct_memory(current->mm,
+       pages * VM_ACCT(PAGE_CACHE_SIZE));
+ vm_unacct_memory(current->mm, pages * VM_ACCT(PAGE_CACHE_SIZE));
 }

 static const struct super_operations shmem_ops;
diff --git a/mm/swapfile.c b/mm/swapfile.c
index bd1bb59..87b7d1a 100644
--- a/mm/swapfile.c
+++ b/mm/swapfile.c
@@ -1213,7 +1213,7 @@ asmlinkage long sys_swapoff(const char __user * specialfile)
  char * pathname;
  int i, type, prev;
  int err;
-
+
  if (!capable(CAP_SYS_ADMIN))
   return -EPERM;

@@ -1245,7 +1245,7 @@ asmlinkage long sys_swapoff(const char __user * specialfile)
   goto out_dput;
  }
  if (!security_vm_enough_memory(p->pages))
-  vm_unacct_memory(p->pages);
+  vm_unacct_memory(current->mm, p->pages);
  else {
   err = -ENOMEM;
```

```
  spin_unlock(&swap_lock);
```

--
1.5.4.3