Subject: [RFC][PATCH][2/4] Add RSS accounting and control (
Posted by Balbir Singh on Sat, 24 Feb 2007 14:45:24 GMT
View Forum Message <> Reply to Message

Changelog

1. Be consistent, use the C style of returning 0 on success and negative
   values on failure
2. Change and document the locking used by the controller
   (I hope I got it right this time :-))
3. Remove memctlr_double_(un)lock routines
4. Comment the usage of MEMCONTROL_DONT_CHECK_LIMIT

This patch adds the basic accounting hooks to account for pages allocated
into the RSS of a process. Accounting is maintained at two levels, in
the mm_struct of each task and in the memory controller data structure
associated with each node in the container.

When the limit specified for the container is exceeded, the task is killed.
RSS accounting is consistent with the current definition of RSS in the
kernel. Shared pages are accounted into the RSS of each process as is
done in the kernel currently. The code is flexible in that it can be easily
modified to work with any definition of RSS.

Signed-off-by: <balbir@in.ibm.com>
---

 fs/exec.c | 4 +
 include/linux/memcontrol.h | 47 ++++++++++++++++
 include/linux/sched.h | 11 +++
 kernel/fork.c | 10 +++
 mm/memcontrol.c | 130 ++---
 mm/memory.c | 34 ++++++++++-
 mm/rmap.c | 5 +
 mm/swapfile.c | 2
 8 files changed, 234 insertions(+), 9 deletions(-)

diff -puN fs/exec.c~memcontrol-acct fs/exec.c
--- linux-2.6.20/fs/exec.c~memcontrol-acct 2007-02-24 19:39:29.000000000 +0530
+++ linux-2.6.20-balbir/fs/exec.c 2007-02-24 19:39:29.000000000 +0530
@@ -50,6 +50,7 @@
 #include <linux/tsacct_kern.h>
 #include <linux/cn_proc.h>
 #include <linux/audit.h>
+#include <linux/memcontrol.h>

 #include <asm/uaccess.h>

```
 #include <asm/mmu_context.h>
@@ -313,6 +314,9 @@ void install_arg_page(struct vm_area_str
  if (unlikely(anon_vma_prepare(vma)))
   goto out;

+ if (memcontrol_update_rss(mm, 1, MEMCONTROL_CHECK_LIMIT))
+  goto out;
+
  flush_dcache_page(page);
  pte = get_locked_pte(mm, address, &ptl);
  if (!pte)
diff -puN include/linux/memcontrol.h~memcontrol-acct include/linux/memcontrol.h
--- linux-2.6.20/include/linux/memcontrol.h~memcontrol-acct 2007-02-24 19:39:29.000000000
+0530
+++ linux-2.6.20-balbir/include/linux/memcontrol.h 2007-02-24 19:39:29.000000000 +0530
@@ -22,12 +22,59 @@
 #ifndef _LINUX_MEMCONTROL_H
 #define _LINUX_MEMCONTROL_H

+/*
+ * MEMCONTROL_DONT_CHECK_LIMIT is useful for the following cases
+ * 1. During fork(), since pages are shared COW, we don't enforce limits
+ *    on fork
+ * 2. During zeromap_pte_range(), again we don't enforce the limit for
+ *    sharing ZERO_PAGE() in this case
+ * 3. When we actually reduce the RSS, add -1 to the rss
+ * It is generally useful when we do not want to enforce limits
+ */
+enum {
+ MEMCONTROL_CHECK_LIMIT = true,
+ MEMCONTROL_DONT_CHECK_LIMIT = false,
+};
+
 #ifdef CONFIG_CONTAINER_MEMCONTROL
+
 #ifndef kB
 #define kB 1024   /* One Kilo Byte */
 #endif

+struct res_counter {
+ atomic_long_t usage; /* The current usage of the resource being */
+    /* counted        */
+ atomic_long_t limit; /* The limit on the resource     */
+};
+
+extern int memcontrol_mm_init(struct mm_struct *mm);
+extern void memcontrol_mm_free(struct mm_struct *mm);
+extern void memcontrol_mm_assign_container(struct mm_struct *mm,
```

```
+      struct task_struct *p);
+extern int memcontrol_update_rss(struct mm_struct *mm, int count, bool check);
+
 #else /* CONFIG_CONTAINER_MEMCONTROL  */

+static inline int memcontrol_mm_init(struct mm_struct *mm)
+{
+ return 0;
+}
+
+static inline void memcontrol_mm_free(struct mm_struct *mm)
+{
+}
+
+static inline void memcontrol_mm_assign_container(struct mm_struct *mm,
+       struct task_struct *p)
+{
+}
+
+static inline int memcontrol_update_rss(struct mm_struct *mm, int count,
+     bool check)
+{
+ return 0;
+}
+
 #endif /* CONFIG_CONTAINER_MEMCONTROL */
 #endif /* _LINUX_MEMCONTROL_H */
diff -puN include/linux/sched.h~memcontrol-acct include/linux/sched.h
--- linux-2.6.20/include/linux/sched.h~memcontrol-acct 2007-02-24 19:39:29.000000000 +0530
+++ linux-2.6.20-balbir/include/linux/sched.h 2007-02-24 19:39:29.000000000 +0530
@@ -83,6 +83,7 @@ struct sched_param {
 #include <linux/timer.h>
 #include <linux/hrtimer.h>
 #include <linux/task_io_accounting.h>
+#include <linux/memcontrol.h>

 #include <asm/processor.h>

@@ -373,6 +374,16 @@ struct mm_struct {
  /* aio bits */
  rwlock_t  ioctx_list_lock;
  struct kioctx  *ioctx_list;
+#ifdef CONFIG_CONTAINER_MEMCONTROL
+ /*
+  * Each mm_struct's container, sums up in the container's counter
+  * We can extend this such that, VMA's counters sum up into this
+  * counter
+  */
```

```
+ struct res_counter *counter;
+ struct container *container;
+ rwlock_t   container_lock;
+#endif /* CONFIG_CONTAINER_MEMCONTROL */
 };

 struct sighand_struct {
diff -puN kernel/fork.c~memcontrol-acct kernel/fork.c
--- linux-2.6.20/kernel/fork.c~memcontrol-acct 2007-02-24 19:39:29.000000000 +0530
+++ linux-2.6.20-balbir/kernel/fork.c 2007-02-24 19:39:29.000000000 +0530
@@ -50,6 +50,7 @@
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>
 #include <linux/numtasks.h>
+#include <linux/memcontrol.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -342,10 +343,15 @@ static struct mm_struct * mm_init(struct
  mm->free_area_cache = TASK_UNMAPPED_BASE;
  mm->cached_hole_size = ~0UL;

+ if (memcontrol_mm_init(mm))
+  goto err;
+
  if (likely(!mm_alloc_pgd(mm))) {
   mm->def_flags = 0;
   return mm;
  }
+
+err:
  free_mm(mm);
  return NULL;
 }
@@ -361,6 +367,8 @@ struct mm_struct * mm_alloc(void)
  if (mm) {
   memset(mm, 0, sizeof(*mm));
   mm = mm_init(mm);
+  if (mm)
+   memcontrol_mm_assign_container(mm, current);
  }
  return mm;
 }
@@ -375,6 +383,7 @@ void fastcall __mmdrop(struct mm_struct
  BUG_ON(mm == &init_mm);
  mm_free_pgd(mm);
  destroy_context(mm);
+ memcontrol_mm_free(mm);
```

```
  free_mm(mm);
 }

@@ -500,6 +509,7 @@ static struct mm_struct *dup_mm(struct t
  if (init_new_context(tsk, mm))
   goto fail_nocontext;

+ memcontrol_mm_assign_container(mm, tsk);
  err = dup_mmap(mm, oldmm);
  if (err)
   goto free_pt;
diff -puN mm/memcontrol.c~memcontrol-acct mm/memcontrol.c
--- linux-2.6.20/mm/memcontrol.c~memcontrol-acct 2007-02-24 19:39:29.000000000 +0530
+++ linux-2.6.20-balbir/mm/memcontrol.c 2007-02-24 19:40:35.000000000 +0530
@@ -30,11 +30,20 @@
 #define RES_USAGE_NO_LIMIT 0
 static const char version[] = "0.1";

-struct res_counter {
- atomic_long_t usage; /* The current usage of the resource being */
-    /* counted        */
- atomic_long_t limit; /* The limit on the resource     */
-};
+/*
+ * Locking notes
+ *
+ * Each mm_struct belongs to a container, when the thread group leader
+ * moves from one container to another, the mm_struct's container is
+ * also migrated to the new container.
+ *
+ * We use a reader/writer lock for consistency. All changes to mm->container
+ * are protected by mm->container_lock. The lock prevents the use-after-free
+ * race condition that can occur. Without the lock, the mm->container
+ * pointer could change from under us.
+ *
+ * The counter uses atomic operations and does not require locking.
+ */

 /*
  * Each task belongs to a container, each container has a struct
@@ -58,11 +67,76 @@ static inline struct memcontrol *memcont
  return memcontrol_from_cont(task_container(p, &memcontrol_subsys));
 }

+int memcontrol_mm_init(struct mm_struct *mm)
+{
+ mm->counter = kmalloc(sizeof(struct res_counter), GFP_KERNEL);
+ if (mm->counter == NULL)
```

```
+  return -ENOMEM;
+ atomic_long_set(&mm->counter->usage, 0);
+ atomic_long_set(&mm->counter->limit, 0);
+ rwlock_init(&mm->container_lock);
+ return 0;
+}
+
+void memcontrol_mm_free(struct mm_struct *mm)
+{
+ kfree(mm->counter);
+}
+
+static inline void memcontrol_mm_assign_container_direct(struct mm_struct *mm,
+       struct container *cont)
+{
+ write_lock(&mm->container_lock);
+ mm->container = cont;
+ write_unlock(&mm->container_lock);
+}
+
+void memcontrol_mm_assign_container(struct mm_struct *mm, struct task_struct *p)
+{
+ struct container *cont = task_container(p, &memcontrol_subsys);
+
+ BUG_ON(!cont);
+ memcontrol_mm_assign_container_direct(mm, cont);
+}
+
+/*
+ * Update the rss usage counters for the mm_struct and the container it belongs
+ * to. We do not fail rss for pages shared during fork (see copy_one_pte()).
+ */
+int memcontrol_update_rss(struct mm_struct *mm, int count, bool check)
+{
+ int ret = 0;
+ struct container *cont;
+ long usage, limit;
+ struct memcontrol *mem;
+
+ read_lock(&mm->container_lock);
+ cont = mm->container;
+
+ if (cont == NULL)
+  goto out_unlock;
+
+ mem = memcontrol_from_cont(cont);
+ usage = atomic_long_read(&mem->counter.usage);
+ limit = atomic_long_read(&mem->counter.limit);
```

```
+ usage += count;
+ if (check && limit && (usage > limit))
+  ret = -ENOMEM; /* Above limit, fail */
+ else {
+  atomic_long_add(count, &mem->counter.usage);
+  atomic_long_add(count, &mm->counter->usage);
+ }
+
+out_unlock:
+ read_unlock(&mm->container_lock);
+ return ret;
+}
+
 static int memcontrol_create(struct container_subsys *ss,
     struct container *cont)
 {
  struct memcontrol *mem = kzalloc(sizeof(*mem), GFP_KERNEL);
- if (!mem)
+ if (mem == NULL)
   return -ENOMEM;

  cont->subsys[memcontrol_subsys.subsys_id] = &mem->css;
@@ -174,11 +248,55 @@ static int memcontrol_populate(struct co
  return 0;
 }

+int memcontrol_can_attach(struct container_subsys *ss, struct container *cont,
+   struct task_struct *p)
+{
+ /*
+  * Allow only the thread group leader to change containers
+  */
+ if (p->pid != p->tgid)
+  return -EINVAL;
+ return 0;
+}
+
+/*
+ * This routine decides how task movement across containers is handled
+ * The simplest strategy is to just move the task (without carrying any old
+ * baggage) The other possibility is move over last accounting information
+ * from mm_struct and charge the new container. We implement the latter.
+ */
+static void memcontrol_attach(struct container_subsys *ss,
+    struct container *cont,
+    struct container *old_cont,
+    struct task_struct *p)
+{
```

```
+ struct memcontrol *mem, *old_mem;
+ long usage;
+
+ if (cont == old_cont)
+  return;
+
+ mem = memcontrol_from_cont(cont);
+ old_mem = memcontrol_from_cont(old_cont);
+
+ memcontrol_mm_assign_container_direct(p->mm, cont);
+ usage = atomic_read(&p->mm->counter->usage);
+ /*
+  * NOTE: we do not fail the movement in case the addition of a new
+  * task, puts the container overlimit. We reclaim and try our best
+  * to push back the usage of the container.
+  */
+ atomic_long_add(usage, &mem->counter.usage);
+ atomic_long_sub(usage, &old_mem->counter.usage);
+}
+
 static struct container_subsys memcontrol_subsys = {
  .name = "memcontrol",
  .create = memcontrol_create,
  .destroy = memcontrol_destroy,
  .populate = memcontrol_populate,
+ .attach = memcontrol_attach,
+ .can_attach = memcontrol_can_attach,
 };

 int __init memcontrol_init(void)
diff -puN mm/memory.c~memcontrol-acct mm/memory.c
--- linux-2.6.20/mm/memory.c~memcontrol-acct 2007-02-24 19:39:29.000000000 +0530
+++ linux-2.6.20-balbir/mm/memory.c 2007-02-24 19:39:29.000000000 +0530
@@ -50,6 +50,7 @@
 #include <linux/delayacct.h>
 #include <linux/init.h>
 #include <linux/writeback.h>
+#include <linux/memcontrol.h>

 #include <asm/pgalloc.h>
 #include <asm/uaccess.h>
@@ -532,6 +533,8 @@ again:
  spin_unlock(src_ptl);
  pte_unmap_nested(src_pte - 1);
  add_mm_rss(dst_mm, rss[0], rss[1]);
+ memcontrol_update_rss(dst_mm, rss[0] + rss[1],
+    MEMCONTROL_DONT_CHECK_LIMIT);
  pte_unmap_unlock(dst_pte - 1, dst_ptl);
```

```
  cond_resched();
  if (addr != end)
@@ -1128,6 +1131,7 @@ static int zeromap_pte_range(struct mm_s
    page_cache_get(page);
    page_add_file_rmap(page);
    inc_mm_counter(mm, file_rss);
+   memcontrol_update_rss(mm, 1, MEMCONTROL_DONT_CHECK_LIMIT);
    set_pte_at(mm, addr, pte, zero_pte);
   } while (pte++, addr += PAGE_SIZE, addr != end);
   arch_leave_lazy_mmu_mode();
@@ -1223,6 +1227,10 @@ static int insert_page(struct mm_struct
   if (PageAnon(page))
    goto out;
   retval = -ENOMEM;
+
+ if (memcontrol_update_rss(mm, 1, MEMCONTROL_CHECK_LIMIT))
+   goto out;
+
   flush_dcache_page(page);
   pte = get_locked_pte(mm, addr, &ptl);
   if (!pte)
@@ -1580,6 +1588,9 @@ gotten:
    cow_user_page(new_page, old_page, address, vma);
   }

+ if (memcontrol_update_rss(mm, 1, MEMCONTROL_CHECK_LIMIT))
+   goto oom;
+
   /*
    * Re-check the pte - we dropped the lock
    */
@@ -1612,7 +1623,9 @@ gotten:
    /* Free the old page.. */
    new_page = old_page;
    ret |= VM_FAULT_WRITE;
- }
+ } else
+   memcontrol_update_rss(mm, -1, MEMCONTROL_DONT_CHECK_LIMIT);
+
   if (new_page)
    page_cache_release(new_page);
   if (old_page)
@@ -2024,16 +2037,19 @@ static int do_swap_page(struct mm_struct
   mark_page_accessed(page);
   lock_page(page);

+ if (memcontrol_update_rss(mm, 1, MEMCONTROL_CHECK_LIMIT))
+   goto out_nomap;
```

```
+
  /*
   * Back out if somebody else already faulted in this pte.
   */
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
  if (unlikely(!pte_same(*page_table, orig_pte)))
-  goto out_nomap;
+  goto out_nomap_uncharge;

  if (unlikely(!PageUptodate(page))) {
   ret = VM_FAULT_SIGBUS;
-  goto out_nomap;
+  goto out_nomap_uncharge;
  }

  /* The page isn't present yet, go ahead with the fault. */
@@ -2068,6 +2084,8 @@ unlock:
  pte_unmap_unlock(page_table, ptl);
 out:
  return ret;
+out_nomap_uncharge:
+ memcontrol_update_rss(mm, -1, MEMCONTROL_DONT_CHECK_LIMIT);
 out_nomap:
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
@@ -2092,6 +2110,9 @@ static int do_anonymous_page(struct mm_s
   /* Allocate our own private page. */
   pte_unmap(page_table);

+  if (memcontrol_update_rss(mm, 1, MEMCONTROL_CHECK_LIMIT))
+   goto oom;
+
   if (unlikely(anon_vma_prepare(vma)))
    goto oom;
   page = alloc_zeroed_user_highpage(vma, address);
@@ -2108,6 +2129,8 @@ static int do_anonymous_page(struct mm_s
   lru_cache_add_active(page);
   page_add_new_anon_rmap(page, vma, address);
  } else {
+  memcontrol_update_rss(mm, 1, MEMCONTROL_DONT_CHECK_LIMIT);
+
   /* Map the ZERO_PAGE - vm_page_prot is readonly */
   page = ZERO_PAGE(address);
   page_cache_get(page);
@@ -2218,6 +2241,9 @@ retry:
   }
  }
```

```
+ if (memcontrol_update_rss(mm, 1, MEMCONTROL_CHECK_LIMIT))
+  goto oom;
+
  page_table = pte_offset_map_lock(mm, pmd, address, &ptl);
  /*
   * For a file-backed vma, someone could have truncated or otherwise
@@ -2227,6 +2253,7 @@ retry:
  if (mapping && unlikely(sequence != mapping->truncate_count)) {
   pte_unmap_unlock(page_table, ptl);
   page_cache_release(new_page);
+  memcontrol_update_rss(mm, -1, MEMCONTROL_DONT_CHECK_LIMIT);
   cond_resched();
   sequence = mapping->truncate_count;
   smp_rmb();
@@ -2265,6 +2292,7 @@ retry:
  } else {
   /* One of our sibling threads was faster, back out. */
   page_cache_release(new_page);
+  memcontrol_update_rss(mm, -1, MEMCONTROL_DONT_CHECK_LIMIT);
   goto unlock;
  }

diff -puN mm/rmap.c~memcontrol-acct mm/rmap.c
--- linux-2.6.20/mm/rmap.c~memcontrol-acct 2007-02-24 19:39:29.000000000 +0530
+++ linux-2.6.20-balbir/mm/rmap.c 2007-02-24 19:39:29.000000000 +0530
@@ -602,6 +602,11 @@ void page_remove_rmap(struct page *page,
   __dec_zone_page_state(page,
     PageAnon(page) ? NR_ANON_PAGES : NR_FILE_MAPPED);
  }
+ /*
+  * When we pass MEMCONTROL_DONT_CHECK_LIMIT, it is ok to call
+  * this function under the pte lock (since we will not block in reclaim)
+  */
+ memcontrol_update_rss(vma->vm_mm, -1, MEMCONTROL_DONT_CHECK_LIMIT);
 }

 /*
diff -puN mm/swapfile.c~memcontrol-acct mm/swapfile.c
--- linux-2.6.20/mm/swapfile.c~memcontrol-acct 2007-02-24 19:39:29.000000000 +0530
+++ linux-2.6.20-balbir/mm/swapfile.c 2007-02-24 19:39:29.000000000 +0530
@@ -27,6 +27,7 @@
 #include <linux/mutex.h>
 #include <linux/capability.h>
 #include <linux/syscalls.h>
+#include <linux/memcontrol.h>

 #include <asm/pgtable.h>
 #include <asm/tlbflush.h>
```

```
@@ -514,6 +515,7 @@ static void unuse_pte(struct vm_area_str
  set_pte_at(vma->vm_mm, addr, pte,
     pte_mkold(mk_pte(page, vma->vm_page_prot)));
  page_add_anon_rmap(page, vma, addr);
+ memcontrol_update_rss(vma->vm_mm, 1, MEMCONTROL_DONT_CHECK_LIMIT);
  swap_free(entry);
  /*
   * Move the page to the active list so it is not
```

_

--
Warm Regards,
Balbir Singh