Subject: [RFC][PATCH][2/4] Add RSS accounting and control
Posted by Balbir Singh on Mon, 19 Feb 2007 06:50:34 GMT
View Forum Message <> Reply to Message

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/memctlr.h |   38 ++++++++++++
 include/linux/sched.h   |   11 +++
 kernel/fork.c           |   10 +++
 mm/memctlr.c            |  148 ++++++++++++++++++++++++++++++++++++++++++------
 mm/memory.c             |   33 +++++++++-
 mm/rmap.c               |    5 +
 mm/swapfile.c           |    2 
 8 files changed, 232 insertions(+), 19 deletions(-)

diff -puN fs/exec.c~memctlr-acct fs/exec.c
--- linux-2.6.20/fs/exec.c~memctlr-acct 2007-02-18 22:55:50.000000000 +0530
+++ linux-2.6.20-balbir/fs/exec.c 2007-02-18 22:55:50.000000000 +0530
@@ -50,6 +50,7 @@
 #include <linux/tsacct_kern.h>
 #include <linux/cn_proc.h>
 #include <linux/audit.h>
+#include <linux/memctlr.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 (!memctlr_update_rss(mm, 1, MEMCTLR_CHECK_LIMIT))
+  goto out;
+
  flush_dcache_page(page);
  pte = get_locked_pte(mm, address, &ptl);
```

```
  if (!pte)
diff -puN include/linux/memctlr.h~memctlr-acct include/linux/memctlr.h
--- linux-2.6.20/include/linux/memctlr.h~memctlr-acct 2007-02-18 22:55:50.000000000 +0530
+++ linux-2.6.20-balbir/include/linux/memctlr.h 2007-02-18 23:28:16.000000000 +0530
@@ -14,9 +14,47 @@
 #ifndef _LINUX_MEMCTLR_H
 #define _LINUX_MEMCTLR_H

+enum {
+ MEMCTLR_CHECK_LIMIT = true,
+ MEMCTLR_DONT_CHECK_LIMIT = false,
+};
+
 #ifdef CONFIG_CONTAINER_MEMCTLR

+struct res_counter {
+ atomic_long_t usage; /* The current usage of the resource being */
+    /* counted        */
+ atomic_long_t limit; /* The limit on the resource     */
+ atomic_long_t nr_limit_exceeded;
+};
+
+extern int memctlr_mm_init(struct mm_struct *mm);
+extern void memctlr_mm_free(struct mm_struct *mm);
+extern void memctlr_mm_assign_container(struct mm_struct *mm,
+      struct task_struct *p);
+extern int memctlr_update_rss(struct mm_struct *mm, int count, bool check);
+
 #else /* CONFIG_CONTAINER_MEMCTLR  */

+static inline int memctlr_mm_init(struct mm_struct *mm)
+{
+ return 0;
+}
+
+static inline void memctlr_mm_free(struct mm_struct *mm)
+{
+}
+
+static inline void memctlr_mm_assign_container(struct mm_struct *mm,
+      struct task_struct *p)
+{
+}
+
+static inline int memctlr_update_rss(struct mm_struct *mm, int count,
+     bool check)
+{
+ return 0;
```

```
+}
+
 #endif /* CONFIG_CONTAINER_MEMCTLR */
 #endif /* _LINUX_MEMCTLR_H */
diff -puN include/linux/sched.h~memctlr-acct include/linux/sched.h
--- linux-2.6.20/include/linux/sched.h~memctlr-acct 2007-02-18 22:55:50.000000000 +0530
+++ linux-2.6.20-balbir/include/linux/sched.h 2007-02-18 22:57:03.000000000 +0530
@@ -83,6 +83,7 @@ struct sched_param {
 #include <linux/timer.h>
 #include <linux/hrtimer.h>
 #include <linux/task_io_accounting.h>
+#include <linux/memctlr.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_MEMCTLR
+ /*
+  * 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_MEMCTLR */
 };

 struct sighand_struct {
diff -puN kernel/fork.c~memctlr-acct kernel/fork.c
--- linux-2.6.20/kernel/fork.c~memctlr-acct 2007-02-18 22:55:50.000000000 +0530
+++ linux-2.6.20-balbir/kernel/fork.c 2007-02-18 22:55:50.000000000 +0530
@@ -50,6 +50,7 @@
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>
 #include <linux/numtasks.h>
+#include <linux/memctlr.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 (!memctlr_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)
+   memctlr_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);
+ memctlr_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;

+ memctlr_mm_assign_container(mm, tsk);
  err = dup_mmap(mm, oldmm);
  if (err)
   goto free_pt;
diff -puN mm/memctlr.c~memctlr-acct mm/memctlr.c
--- linux-2.6.20/mm/memctlr.c~memctlr-acct 2007-02-18 22:55:50.000000000 +0530
+++ linux-2.6.20-balbir/mm/memctlr.c 2007-02-18 23:29:09.000000000 +0530
@@ -23,13 +23,6 @@
 #define RES_USAGE_NO_LIMIT   0
 static const char version[] = "0.1";

-struct res_counter {
- unsigned long usage; /* The current usage of the resource being */
-    /* counted        */
- unsigned long limit; /* The limit on the resource     */
- unsigned long nr_limit_exceeded;
```

```
-};
-
 struct memctlr {
  struct container_subsys_state css;
  struct res_counter   counter;
@@ -49,6 +42,74 @@ static inline struct memctlr *memctlr_fr
  return memctlr_from_cont(task_container(p, &memctlr_subsys));
 }

+int memctlr_mm_init(struct mm_struct *mm)
+{
+ mm->counter = kmalloc(sizeof(struct res_counter), GFP_KERNEL);
+ if (!mm->counter)
+  return 0;
+ atomic_long_set(&mm->counter->usage, 0);
+ atomic_long_set(&mm->counter->limit, 0);
+ rwlock_init(&mm->container_lock);
+ return 1;
+}
+
+void memctlr_mm_free(struct mm_struct *mm)
+{
+ kfree(mm->counter);
+}
+
+static inline void memctlr_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 memctlr_mm_assign_container(struct mm_struct *mm, struct task_struct *p)
+{
+ struct container *cont = task_container(p, &memctlr_subsys);
+ struct memctlr *mem = memctlr_from_cont(cont);
+
+ BUG_ON(!mem);
+ write_lock(&mm->container_lock);
+ mm->container = cont;
+ write_unlock(&mm->container_lock);
+}
+
+/*
+ * 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 memctlr_update_rss(struct mm_struct *mm, int count, bool check)
+{
+ int ret = 1;
+ struct container *cont;
+ long usage, limit;
+ struct memctlr *mem;
+
+ read_lock(&mm->container_lock);
+ cont = mm->container;
+ read_unlock(&mm->container_lock);
+
+ if (!cont)
+  goto done;
+
+ mem = memctlr_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 = 0; /* Above limit, fail */
+ else {
+  atomic_long_add(count, &mem->counter.usage);
+  atomic_long_add(count, &mm->counter->usage);
+ }
+
+done:
+ return ret;
+}
+
 static int memctlr_create(struct container_subsys *ss, struct container *cont)
 {
  struct memctlr *mem = kzalloc(sizeof(*mem), GFP_KERNEL);
@@ -57,6 +118,8 @@ static int memctlr_create(struct contain

  spin_lock_init(&mem->lock);
  cont->subsys[memctlr_subsys.subsys_id] = &mem->css;
+ atomic_long_set(&mem->counter.usage, 0);
+ atomic_long_set(&mem->counter.limit, 0);
  return 0;
 }

@@ -98,9 +161,7 @@ static ssize_t memctlr_write(struct cont
  if (!limit && strcmp(buffer, "0"))
   goto out_unlock;

- spin_lock(&mem->lock);
- mem->counter.limit = limit;
- spin_unlock(&mem->lock);
```

```
+ atomic_long_set(&mem->counter.limit, limit);

  ret = nbytes;
 out_unlock:
@@ -114,17 +175,15 @@ static ssize_t memctlr_read(struct conta
     struct file *file, char __user *userbuf,
     size_t nbytes, loff_t *ppos)
 {
- unsigned long usage, limit;
+ long usage, limit;
  char usagebuf[64];  /* Move away from stack later */
  char *s = usagebuf;
  struct memctlr *mem = memctlr_from_cont(cont);

- spin_lock(&mem->lock);
- usage = mem->counter.usage;
- limit = mem->counter.limit;
- spin_unlock(&mem->lock);
+ usage = atomic_long_read(&mem->counter.usage);
+ limit = atomic_long_read(&mem->counter.limit);

- s += sprintf(s, "usage %lu, limit %ld\n", usage, limit);
+ s += sprintf(s, "usage %ld, limit %ld\n", usage, limit);
  return simple_read_from_buffer(userbuf, nbytes, ppos, usagebuf,
      s - usagebuf);
 }
@@ -150,11 +209,68 @@ static int memctlr_populate(struct conta
  return 0;
 }

+static inline void memctlr_double_lock(struct memctlr *mem1,
+      struct memctlr *mem2)
+{
+ if (mem1 > mem2) {
+  spin_lock(&mem1->lock);
+  spin_lock(&mem2->lock);
+ } else {
+  spin_lock(&mem2->lock);
+  spin_lock(&mem1->lock);
+ }
+}
+
+static inline void memctlr_double_unlock(struct memctlr *mem1,
+       struct memctlr *mem2)
+{
+ if (mem1 > mem2) {
+  spin_unlock(&mem2->lock);
+  spin_unlock(&mem1->lock);
```

```
+ } else {
+  spin_unlock(&mem1->lock);
+  spin_unlock(&mem2->lock);
+ }
+}
+
+/*
+ * 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
+ */
+static void memctlr_attach(struct container_subsys *ss,
+    struct container *cont,
+    struct container *old_cont,
+    struct task_struct *p)
+{
+ struct memctlr *mem, *old_mem;
+ long usage;
+
+ mem = memctlr_from_cont(cont);
+ old_mem = memctlr_from_cont(old_cont);
+
+ memctlr_double_lock(mem, old_mem);
+
+ memctlr_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);
+
+ memctlr_double_unlock(mem, old_mem);
+}
+
 static struct container_subsys memctlr_subsys = {
  .name = "memctlr",
  .create = memctlr_create,
  .destroy = memctlr_destroy,
  .populate = memctlr_populate,
+ .attach = memctlr_attach,
 };

 int __init memctlr_init(void)
diff -puN mm/memory.c~memctlr-acct mm/memory.c
```

```
--- linux-2.6.20/mm/memory.c~memctlr-acct 2007-02-18 22:55:50.000000000 +0530
+++ linux-2.6.20-balbir/mm/memory.c 2007-02-18 22:55:50.000000000 +0530
@@ -50,6 +50,7 @@
 #include <linux/delayacct.h>
 #include <linux/init.h>
 #include <linux/writeback.h>
+#include <linux/memctlr.h>

 #include <asm/pgalloc.h>
 #include <asm/uaccess.h>
@@ -532,6 +533,7 @@ again:
  spin_unlock(src_ptl);
  pte_unmap_nested(src_pte - 1);
  add_mm_rss(dst_mm, rss[0], rss[1]);
+ memctlr_update_rss(dst_mm, rss[0] + rss[1], MEMCTLR_DONT_CHECK_LIMIT);
  pte_unmap_unlock(dst_pte - 1, dst_ptl);
  cond_resched();
  if (addr != end)
@@ -1128,6 +1130,7 @@ static int zeromap_pte_range(struct mm_s
   page_cache_get(page);
   page_add_file_rmap(page);
   inc_mm_counter(mm, file_rss);
+  memctlr_update_rss(mm, 1, MEMCTLR_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 +1226,10 @@ static int insert_page(struct mm_struct
  if (PageAnon(page))
   goto out;
  retval = -ENOMEM;
+
+ if (!memctlr_update_rss(mm, 1, MEMCTLR_CHECK_LIMIT))
+  goto out;
+
  flush_dcache_page(page);
  pte = get_locked_pte(mm, addr, &ptl);
  if (!pte)
@@ -1580,6 +1587,9 @@ gotten:
   cow_user_page(new_page, old_page, address, vma);
  }

+ if (!memctlr_update_rss(mm, 1, MEMCTLR_CHECK_LIMIT))
+  goto oom;
+
  /*
   * Re-check the pte - we dropped the lock
   */
@@ -1612,7 +1622,9 @@ gotten:
```

```
   /* Free the old page.. */
   new_page = old_page;
   ret |= VM_FAULT_WRITE;
- }
+ } else
+  memctlr_update_rss(mm, -1, MEMCTLR_DONT_CHECK_LIMIT);
+
  if (new_page)
   page_cache_release(new_page);
  if (old_page)
@@ -2024,16 +2036,19 @@ static int do_swap_page(struct mm_struct
  mark_page_accessed(page);
  lock_page(page);

+ if (!memctlr_update_rss(mm, 1, MEMCTLR_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 +2083,8 @@ unlock:
  pte_unmap_unlock(page_table, ptl);
 out:
  return ret;
+out_nomap_uncharge:
+ memctlr_update_rss(mm, -1, MEMCTLR_DONT_CHECK_LIMIT);
 out_nomap:
  pte_unmap_unlock(page_table, ptl);
  unlock_page(page);
@@ -2092,6 +2109,9 @@ static int do_anonymous_page(struct mm_s
   /* Allocate our own private page. */
   pte_unmap(page_table);

+  if (!memctlr_update_rss(mm, 1, MEMCTLR_CHECK_LIMIT))
+   goto oom;
+
```

```
   if (unlikely(anon_vma_prepare(vma)))
    goto oom;
   page = alloc_zeroed_user_highpage(vma, address);
@@ -2108,6 +2128,8 @@ static int do_anonymous_page(struct mm_s
   lru_cache_add_active(page);
   page_add_new_anon_rmap(page, vma, address);
  } else {
+  memctlr_update_rss(mm, 1, MEMCTLR_DONT_CHECK_LIMIT);
+
   /* Map the ZERO_PAGE - vm_page_prot is readonly */
   page = ZERO_PAGE(address);
   page_cache_get(page);
@@ -2218,6 +2240,9 @@ retry:
   }
  }

+ if (!memctlr_update_rss(mm, 1, MEMCTLR_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 +2252,7 @@ retry:
  if (mapping && unlikely(sequence != mapping->truncate_count)) {
   pte_unmap_unlock(page_table, ptl);
   page_cache_release(new_page);
+  memctlr_update_rss(mm, -1, MEMCTLR_DONT_CHECK_LIMIT);
   cond_resched();
   sequence = mapping->truncate_count;
   smp_rmb();
@@ -2265,6 +2291,7 @@ retry:
  } else {
   /* One of our sibling threads was faster, back out. */
   page_cache_release(new_page);
+  memctlr_update_rss(mm, -1, MEMCTLR_DONT_CHECK_LIMIT);
   goto unlock;
  }

diff -puN mm/rmap.c~memctlr-acct mm/rmap.c
--- linux-2.6.20/mm/rmap.c~memctlr-acct 2007-02-18 22:55:50.000000000 +0530
+++ linux-2.6.20-balbir/mm/rmap.c 2007-02-18 23:28:16.000000000 +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 MEMCTLR_DONT_CHECK_LIMIT, it is ok to call
+  * this function under the pte lock (since we will not block in reclaim)
```

```
+  */
+ memctlr_update_rss(vma->vm_mm, -1, MEMCTLR_DONT_CHECK_LIMIT);
 }

 /*
diff -puN mm/swapfile.c~memctlr-acct mm/swapfile.c
--- linux-2.6.20/mm/swapfile.c~memctlr-acct 2007-02-18 22:55:50.000000000 +0530
+++ linux-2.6.20-balbir/mm/swapfile.c 2007-02-18 22:55:50.000000000 +0530
@@ -27,6 +27,7 @@
 #include <linux/mutex.h>
 #include <linux/capability.h>
 #include <linux/syscalls.h>
+#include <linux/memctlr.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);
+ memctlr_update_rss(vma->vm_mm, 1, MEMCTLR_DONT_CHECK_LIMIT);
  swap_free(entry);
  /*
   * Move the page to the active list so it is not
_

--
 Warm Regards,
 Balbir Singh
```