Subject: [RFC PATCH 2/5] overcommit accounting and handling functions
Posted by Andrea Righi on Mon, 09 Jun 2008 23:33:00 GMT
View Forum Message <> Reply to Message

Split the different __vm_enough_memory() policies in inline functions to
easily reuse them in the memory controller overcommit handling routines.

Accounting functions vm_acct_memory() and vm_unacct_memory() are rewritten as
well, including per-cgroup committed VM accounting concept.

Signed-off-by: Andrea Righi <righi.andrea@gmail.com>
---
 include/linux/mman.h | 148 ++--
 mm/memcontrol.c | 139 ++-
 mm/mmap.c | 85 ++++----------------------
 mm/nommu.c | 84 ++++----------------------
 mm/swap.c | 3 +-
 5 files changed, 306 insertions(+), 153 deletions(-)

diff --git a/include/linux/mman.h b/include/linux/mman.h
index dab8892..37f695f 100644
--- a/include/linux/mman.h
+++ b/include/linux/mman.h
@@ -12,25 +12,165 @@

 #ifdef __KERNEL__
 #include <linux/mm.h>
+#include <linux/vmstat.h>
+#include <linux/mmzone.h>
+#include <linux/mm_types.h>
+#include <linux/hugetlb.h>
+#include <linux/swap.h>

 #include <asm/atomic.h>

 extern int sysctl_overcommit_memory;
 extern int sysctl_overcommit_ratio;
 extern atomic_long_t vm_committed_space;
+extern unsigned long totalreserve_pages;
+extern unsigned long totalram_pages;
+
+struct vm_acct_values {
+ int overcommit_memory;
+ int overcommit_ratio;
+ atomic_long_t vm_committed_space;
+};
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR

```
+extern void vm_acct_get_config(const struct mm_struct *mm,
+         struct vm_acct_values *v);
+extern void mem_cgroup_vm_acct_memory(struct mm_struct *mm, long pages);
+#else
+static inline void vm_acct_get_config(const struct mm_struct *mm,
+         struct vm_acct_values *v)
+{
+ v->overcommit_memory = sysctl_overcommit_memory;
+ v->overcommit_ratio = sysctl_overcommit_ratio;
+}
+static inline void mem_cgroup_vm_acct_memory(struct mm_struct *mm, long pages)
+{
+}
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+
+static inline int __vm_enough_memory_guess(struct mm_struct *mm,
+        long pages,
+        int cap_sys_admin)
+{
+ unsigned long n, free;
+
+ free = global_page_state(NR_FILE_PAGES);
+ free += nr_swap_pages;
+
+ /*
+  * Any slabs which are created with the
+  * SLAB_RECLAIM_ACCOUNT flag claim to have contents
+  * which are reclaimable, under pressure.  The dentry
+  * cache and most inode caches should fall into this
+  */
+ free += global_page_state(NR_SLAB_RECLAIMABLE);
+
+ /*
+  * Leave the last 3% for root
+  */
+ if (!cap_sys_admin)
+  free -= free / 32;
+
+ if (free > pages)
+  return 0;
+
+ /*
+  * nr_free_pages() is very expensive on large systems,
+  * only call if we're about to fail.
+  */
+ n = nr_free_pages();
+
+ /*
```

```
+  * Leave reserved pages. The pages are not for anonymous pages.
+  */
+ if (n <= totalreserve_pages)
+  return -ENOMEM;
+ else
+  n -= totalreserve_pages;
+
+ /*
+  * Leave the last 3% for root
+  */
+ if (!cap_sys_admin)
+  n -= n / 32;
+ free += n;
+
+ if (free > pages)
+  return 0;
+
+ return -ENOMEM;
+}
+
+static inline int __vm_enough_memory_never(struct mm_struct *mm,
+        long pages,
+        int cap_sys_admin)
+{
+ unsigned long allowed;
+ struct vm_acct_values v;
+
+ vm_acct_get_config(mm, &v);
+
+ allowed = (totalram_pages - hugetlb_total_pages())
+  * v.overcommit_ratio / 100;
+ /*
+  * Leave the last 3% for root
+  */
+ if (!cap_sys_admin)
+  allowed -= allowed / 32;
+ allowed += total_swap_pages;
+
+ /* Don't let a single process grow too big:
+    leave 3% of the size of this process for other processes */
+ allowed -= mm->total_vm / 32;
+
+ /*
+  * cast `allowed' as a signed long because vm_committed_space
+  * sometimes has a negative value
+  */
+ if (atomic_long_read(&vm_committed_space) < (long)allowed)
+  return 0;
```

```
+
+ return -ENOMEM;
+}
+
+#ifdef CONFIG_CGROUP_MEM_RES_CTLR
+extern int mem_cgroup_vm_enough_memory_guess(struct mm_struct *mm,
+        long pages,
+        int cap_sys_admin);
+
+extern int mem_cgroup_vm_enough_memory_never(struct mm_struct *mm,
+        long pages,
+        int cap_sys_admin);
+#else /* CONFIG_CGROUP_MEM_RES_CTLR */
+static inline int mem_cgroup_vm_enough_memory_guess(struct mm_struct *mm,
+        long pages,
+        int cap_sys_admin)
+{
+ return __vm_enough_memory_guess(mm, pages, cap_sys_admin);
+}
+
+static inline int mem_cgroup_vm_enough_memory_never(struct mm_struct *mm,
+        long pages,
+        int cap_sys_admin)
+{
+ return __vm_enough_memory_never(mm, pages, cap_sys_admin);
+}
+#endif /* CONFIG_CGROUP_MEM_RES_CTLR */
+

 #ifdef CONFIG_SMP
-extern void vm_acct_memory(long pages);
+extern void vm_acct_memory(struct mm_struct *mm, long pages);
 #else
-static inline void vm_acct_memory(long pages)
+static inline void vm_acct_memory(struct mm_struct *mm, long pages)
 {
  atomic_long_add(pages, &vm_committed_space);
+ mem_cgroup_vm_acct_memory(mm, pages);
 }
 #endif

-static inline void vm_unacct_memory(long pages)
+static inline void vm_unacct_memory(struct mm_struct *mm, long pages)
 {
- vm_acct_memory(-pages);
+ vm_acct_memory(mm, -pages);
 }
```

```
 /*
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index e46451e..4100e24 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -21,6 +21,7 @@
 #include <linux/memcontrol.h>
 #include <linux/cgroup.h>
 #include <linux/mm.h>
+#include <linux/mman.h>
 #include <linux/smp.h>
 #include <linux/page-flags.h>
 #include <linux/backing-dev.h>
@@ -141,6 +142,10 @@ struct mem_cgroup {
   * statistics.
   */
  struct mem_cgroup_stat stat;
+ /*
+  * VM overcommit settings
+  */
+ struct vm_acct_values vmacct;
 };
 static struct mem_cgroup init_mem_cgroup;

@@ -187,6 +192,130 @@ enum charge_type {
  MEM_CGROUP_CHARGE_TYPE_MAPPED,
 };

+void vm_acct_get_config(const struct mm_struct *mm, struct vm_acct_values *v)
+{
+ struct mem_cgroup *mem;
+ long tmp;
+
+ BUG_ON(!mm);
+
+ rcu_read_lock();
+ mem = mem_cgroup_from_task(rcu_dereference(mm->owner));
+ v->overcommit_memory = mem->vmacct.overcommit_memory;
+ v->overcommit_ratio = mem->vmacct.overcommit_ratio;
+ tmp = atomic_long_read(&mem->vmacct.vm_committed_space);
+ atomic_long_set(&v->vm_committed_space, tmp);
+ rcu_read_unlock();
+}
+
+void mem_cgroup_vm_acct_memory(struct mm_struct *mm, long pages)
+{
+ struct mem_cgroup *mem;
+ struct task_struct *tsk;
```

```
+
+ if (!mm)
+  return;
+
+ rcu_read_lock();
+ tsk = rcu_dereference(mm->owner);
+ mem = mem_cgroup_from_task(tsk);
+ /* Update memory cgroup statistic */
+ atomic_long_add(pages, &mem->vmacct.vm_committed_space);
+ /* Update task statistic */
+ atomic_long_add(pages, &tsk->vm_committed_space);
+ rcu_read_unlock();
+}
+
+int mem_cgroup_vm_enough_memory_guess(struct mm_struct *mm,
+         long pages,
+         int cap_sys_admin)
+{
+ unsigned long n, free;
+ struct mem_cgroup *mem;
+ long total, rss, cache;
+
+ rcu_read_lock();
+ mem = mem_cgroup_from_task(rcu_dereference(mm->owner));
+ total = (long) (mem->res.limit >> PAGE_SHIFT) + 1L;
+ if (total > (totalram_pages - hugetlb_total_pages())) {
+  rcu_read_unlock();
+  return __vm_enough_memory_guess(mm, pages, cap_sys_admin);
+ }
+ cache = (long)mem_cgroup_read_stat(&mem->stat,
+  MEM_CGROUP_STAT_CACHE);
+ rss = (long)mem_cgroup_read_stat(&mem->stat,
+  MEM_CGROUP_STAT_RSS);
+ rcu_read_unlock();
+
+ free = cache;
+ free += nr_swap_pages;
+
+ /*
+  * Leave the last 3% for root
+  */
+ if (!cap_sys_admin)
+  free -= free / 32;
+
+ if (free > pages)
+  return 0;
+
+ n = total - rss;
```

```
+
+ /*
+  * Leave the last 3% for root
+  */
+ if (!cap_sys_admin)
+  n -= n / 32;
+ free += n;
+
+ if (free > pages)
+  return 0;
+
+ return -ENOMEM;
+}
+
+int mem_cgroup_vm_enough_memory_never(struct mm_struct *mm,
+          long pages,
+          int cap_sys_admin)
+{
+ unsigned long allowed;
+ struct vm_acct_values v;
+ struct mem_cgroup *mem;
+ long total;
+
+ rcu_read_lock();
+ mem = mem_cgroup_from_task(rcu_dereference(mm->owner));
+ total = (long)(mem->res.limit >> PAGE_SHIFT) + 1L;
+ if (total > (totalram_pages - hugetlb_total_pages())) {
+  rcu_read_unlock();
+  return __vm_enough_memory_never(mm, pages, cap_sys_admin);
+ }
+ rcu_read_unlock();
+
+ vm_acct_get_config(mm, &v);
+
+ allowed = total * v.overcommit_ratio / 100;
+ /*
+  * Leave the last 3% for root
+  */
+ if (!cap_sys_admin)
+  allowed -= allowed / 32;
+ allowed += total_swap_pages;
+
+ /* Don't let a single process grow too big:
+    leave 3% of the size of this process for other processes */
+ allowed -= mm->total_vm / 32;
+
+ /*
+  * cast `allowed' as a signed long because vm_committed_space
```

```
+  * sometimes has a negative value
+  */
+ if (atomic_long_read(&v.vm_committed_space) < (long)allowed)
+  return 0;
+
+ return -ENOMEM;
+}
+
 /*
  * Always modified under lru lock. Then, not necessary to preempt_disable()
  */
@@ -1022,17 +1151,25 @@ static struct cgroup_subsys_state *
 mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
 {
  struct mem_cgroup *mem;
+ struct cgroup *p = cont->parent;
  int node;

- if (unlikely((cont->parent) == NULL)) {
+ if (unlikely((p) == NULL)) {
   mem = &init_mem_cgroup;
   page_cgroup_cache = KMEM_CACHE(page_cgroup, SLAB_PANIC);
+  mem->vmacct.overcommit_memory = sysctl_overcommit_memory;
+  mem->vmacct.overcommit_ratio = sysctl_overcommit_ratio;
  } else {
   mem = mem_cgroup_alloc();
   if (!mem)
    return ERR_PTR(-ENOMEM);
+  mem->vmacct.overcommit_memory =
+   mem_cgroup_from_cont(p)->vmacct.overcommit_memory;
+  mem->vmacct.overcommit_ratio =
+   mem_cgroup_from_cont(p)->vmacct.overcommit_ratio;
  }

+ atomic_long_set(&mem->vmacct.vm_committed_space, 0);
  res_counter_init(&mem->res);

  for_each_node_state(node, N_POSSIBLE)
diff --git a/mm/mmap.c b/mm/mmap.c
index 3354fdd..256599e 100644
--- a/mm/mmap.c
+++ b/mm/mmap.c
@@ -25,6 +25,7 @@
 #include <linux/module.h>
 #include <linux/mount.h>
 #include <linux/mempolicy.h>
+#include <linux/memcontrol.h>
 #include <linux/rmap.h>
```

```
 #include <asm/uaccess.h>
@@ -100,87 +101,23 @@ atomic_long_t vm_committed_space = ATOMIC_LONG_INIT(0);
  */
 int __vm_enough_memory(struct mm_struct *mm, long pages, int cap_sys_admin)
 {
- unsigned long free, allowed;
+ struct vm_acct_values v;

- vm_acct_memory(pages);
+ vm_acct_get_config(mm, &v);
+ vm_acct_memory(mm, pages);

  /*
   * Sometimes we want to use more memory than we have
   */
- if (sysctl_overcommit_memory == OVERCOMMIT_ALWAYS)
+ if (v.overcommit_memory == OVERCOMMIT_ALWAYS)
   return 0;
-
- if (sysctl_overcommit_memory == OVERCOMMIT_GUESS) {
-  unsigned long n;
-
-  free = global_page_state(NR_FILE_PAGES);
-  free += nr_swap_pages;
-
-  /*
-   * Any slabs which are created with the
-   * SLAB_RECLAIM_ACCOUNT flag claim to have contents
-   * which are reclaimable, under pressure.  The dentry
-   * cache and most inode caches should fall into this
-   */
-  free += global_page_state(NR_SLAB_RECLAIMABLE);
-
-  /*
-   * Leave the last 3% for root
-   */
-  if (!cap_sys_admin)
-   free -= free / 32;
-
-  if (free > pages)
-   return 0;
-
-  /*
-   * nr_free_pages() is very expensive on large systems,
-   * only call if we're about to fail.
-   */
-  n = nr_free_pages();
```

```
-
-  /*
-   * Leave reserved pages. The pages are not for anonymous pages.
-   */
-  if (n <= totalreserve_pages)
-   goto error;
-  else
-   n -= totalreserve_pages;
-
-  /*
-   * Leave the last 3% for root
-   */
-  if (!cap_sys_admin)
-   n -= n / 32;
-  free += n;
-
-  if (free > pages)
-   return 0;
-
-  goto error;
- }
-
- allowed = (totalram_pages - hugetlb_total_pages())
-         * sysctl_overcommit_ratio / 100;
- /*
-  * Leave the last 3% for root
-  */
- if (!cap_sys_admin)
-  allowed -= allowed / 32;
- allowed += total_swap_pages;
-
- /* Don't let a single process grow too big:
-    leave 3% of the size of this process for other processes */
- allowed -= mm->total_vm / 32;
-
- /*
-  * cast `allowed' as a signed long because vm_committed_space
-  * sometimes has a negative value
-  */
- if (atomic_long_read(&vm_committed_space) < (long)allowed)
+ if ((v.overcommit_memory == OVERCOMMIT_GUESS) &&
+     (!mem_cgroup_vm_enough_memory_guess(mm, pages, cap_sys_admin)))
+  return 0;
+ else if (!mem_cgroup_vm_enough_memory_never(mm, pages, cap_sys_admin))
   return 0;
-error:
- vm_unacct_memory(pages);
+
```

```
+ vm_unacct_memory(mm, pages);

  return -ENOMEM;
 }
diff --git a/mm/nommu.c b/mm/nommu.c
index 3abd084..b194a44 100644
--- a/mm/nommu.c
+++ b/mm/nommu.c
@@ -20,6 +20,7 @@
 #include <linux/file.h>
 #include <linux/highmem.h>
 #include <linux/pagemap.h>
+#include <linux/memcontrol.h>
 #include <linux/slab.h>
 #include <linux/vmalloc.h>
 #include <linux/ptrace.h>
@@ -1356,86 +1357,23 @@ EXPORT_SYMBOL(get_unmapped_area);
  */
 int __vm_enough_memory(struct mm_struct *mm, long pages, int cap_sys_admin)
 {
- unsigned long free, allowed;
+ struct vm_acct_values v;

- vm_acct_memory(pages);
+ vm_acct_get_config(mm, &v);
+ vm_acct_memory(mm, pages);

  /*
   * Sometimes we want to use more memory than we have
   */
- if (sysctl_overcommit_memory == OVERCOMMIT_ALWAYS)
+ if (v.overcommit_memory == OVERCOMMIT_ALWAYS)
   return 0;
-
- if (sysctl_overcommit_memory == OVERCOMMIT_GUESS) {
-  unsigned long n;
-
-  free = global_page_state(NR_FILE_PAGES);
-  free += nr_swap_pages;
-
-  /*
-   * Any slabs which are created with the
-   * SLAB_RECLAIM_ACCOUNT flag claim to have contents
-   * which are reclaimable, under pressure.  The dentry
-   * cache and most inode caches should fall into this
-   */
-  free += global_page_state(NR_SLAB_RECLAIMABLE);
-
```

```
-  /*
-   * Leave the last 3% for root
-   */
-  if (!cap_sys_admin)
-   free -= free / 32;
-
-  if (free > pages)
-   return 0;
-
-  /*
-   * nr_free_pages() is very expensive on large systems,
-   * only call if we're about to fail.
-   */
-  n = nr_free_pages();
-
-  /*
-   * Leave reserved pages. The pages are not for anonymous pages.
-   */
-  if (n <= totalreserve_pages)
-   goto error;
-  else
-   n -= totalreserve_pages;
-
-  /*
-   * Leave the last 3% for root
-   */
-  if (!cap_sys_admin)
-   n -= n / 32;
-  free += n;
-
-  if (free > pages)
-   return 0;
-
-  goto error;
- }
-
- allowed = totalram_pages * sysctl_overcommit_ratio / 100;
- /*
-  * Leave the last 3% for root
-  */
- if (!cap_sys_admin)
-  allowed -= allowed / 32;
- allowed += total_swap_pages;
-
- /* Don't let a single process grow too big:
-    leave 3% of the size of this process for other processes */
- allowed -= current->mm->total_vm / 32;
-
```

```
- /*
-  * cast `allowed' as a signed long because vm_committed_space
-  * sometimes has a negative value
-  */
- if (atomic_long_read(&vm_committed_space) < (long)allowed)
+ if ((v.overcommit_memory == OVERCOMMIT_GUESS) &&
+     (!mem_cgroup_vm_enough_memory_guess(mm, pages, cap_sys_admin)))
+  return 0;
+ else if (!mem_cgroup_vm_enough_memory_never(mm, pages, cap_sys_admin))
    return 0;
-error:
- vm_unacct_memory(pages);
+
+ vm_unacct_memory(mm, pages);

   return -ENOMEM;
  }
diff --git a/mm/swap.c b/mm/swap.c
index 45c9f25..f7676db 100644
--- a/mm/swap.c
+++ b/mm/swap.c
@@ -495,7 +495,7 @@ EXPORT_SYMBOL(pagevec_lookup_tag);

  static DEFINE_PER_CPU(long, committed_space) = 0;

-void vm_acct_memory(long pages)
+void vm_acct_memory(struct mm_struct *mm, long pages)
  {
   long *local;

@@ -507,6 +507,7 @@ void vm_acct_memory(long pages)
    *local = 0;
   }
   preempt_enable();
+ mem_cgroup_vm_acct_memory(mm, pages);
  }

  #ifdef CONFIG_HOTPLUG_CPU
--
1.5.4.3
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers