Subject: Re: Question : memrlimit cgroup's task_move (2.6.26-rc5-mm3)
Posted by Balbir Singh on Thu, 19 Jun 2008 16:41:28 GMT
View Forum Message <> Reply to Message

* KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> [2008-06-19 19:22:27]:

> On Thu, 19 Jun 2008 12:24:29 +0900
> KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com> wrote:
>
> > On Thu, 19 Jun 2008 08:43:43 +0530
> > Balbir Singh <balbir@linux.vnet.ibm.com> wrote:
> >
> > >
> > > > I think the charge of the new group goes to minus. right ?
> > > > (and old group's charge never goes down.)
> > > > I don't think this is "no problem".
> > > >
> > > > What kind of patch is necessary to fix this ?
> > > > task_attach() should be able to fail in future ?
> > > >
> > > > I'm sorry if I misunderstand something or this is already in TODO list.
> > > >
> > >
> > > It's already on the TODO list. Thanks for keeping me reminded about it.
> > >
> > Okay, I'm looking foward to see how can_attach and roll-back(if necessary)
> > is implemnted.
> > As you know, I'm interested in how to handle failure of task move.
> >
> One more thing...
> Now, charge is done at
>
> - vm is inserted (special case?)
> - vm is expanded (mmap is called, stack growth...)
>
> And uncharge is done at
> - vm is removed (success of munmap)
> - exit_mm is called (exit of process)
>
> But it seems charging at may_expand_vm() is not good.
> The mmap can fail after may_expand_vm() because of various reason,
> but charge is already done at may_expand_vm()....and no roll-back.
>
Hi, Kamezawa-San,

The patch I have below addresses the issue. FYI: I do see some
variation in memrlimit.usage_in_bytes after running ls, but it's not
that much. I verified that the patch behaves correctly, by doing

a cat of memrlimit.usage_in_bytes from another shell (not associated
with test group) and then compared that value to /proc/$$/statm
($$ being the pid of the task belonging to the test group).

Could you please review/test the patch below? If it works OK, I'll
request Andrew to pick it up.

Description
----
memrlimit cgroup does not handle error cases after may_expand_vm(). This
BUG was reported by Kamezawa, with the test case below to reproduce it

[root@iridium kamezawa]# cat /opt/cgroup/test/memrlimit.usage_in_bytes
71921664
[root@iridium kamezawa]# ulimit -s 3
[root@iridium kamezawa]# ls
Killed
[root@iridium kamezawa]# ls
Killed
[root@iridium kamezawa]# ls
Killed
[root@iridium kamezawa]# ls
Killed
[root@iridium kamezawa]# ls
Killed
[root@iridium kamezawa]# ulimit -s unlimited
[root@iridium kamezawa]# cat /opt/cgroup/test/memrlimit.usage_in_bytes
72368128
[root@iridium kamezawa]#

This patch adds better handling support to fix the reported problem.

Reported-By: kamezawa.hiroyu@jp.fujitsu.com
Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>
---

 mm/mmap.c | 36 ++++++++++++++++++++++++++----------
 mm/mremap.c | 6 ++++++
 2 files changed, 31 insertions(+), 11 deletions(-)

diff -puN mm/mmap.c~memrlimit-cgroup-add-better-error-handling mm/mmap.c
--- linux-2.6.26-rc5/mm/mmap.c~memrlimit-cgroup-add-better-error-handling 2008-06-19
21:12:46.000000000 +0530
+++ linux-2.6.26-rc5-balbir/mm/mmap.c 2008-06-19 21:39:45.000000000 +0530
@@ -1123,7 +1123,7 @@ munmap_back:
     */
    charged = len >> PAGE_SHIFT;
    if (security_vm_enough_memory(charged))

```
-    return -ENOMEM;
+    goto undo_charge;
     vm_flags |= VM_ACCOUNT;
    }
   }
@@ -1245,6 +1245,8 @@ free_vma:
 unacct_error:
   if (charged)
    vm_unacct_memory(charged);
+undo_charge:
+ memrlimit_cgroup_uncharge_as(mm, len >> PAGE_SHIFT);
   return error;
 }

@@ -1540,14 +1542,15 @@ static int acct_stack_growth(struct vm_a
   struct mm_struct *mm = vma->vm_mm;
   struct rlimit *rlim = current->signal->rlim;
   unsigned long new_start;
+ int ret = -ENOMEM;

   /* address space limit tests */
   if (!may_expand_vm(mm, grow))
-  return -ENOMEM;
+  goto out;

   /* Stack limit test */
   if (size > rlim[RLIMIT_STACK].rlim_cur)
-  return -ENOMEM;
+  goto undo_charge;

   /* mlock limit tests */
   if (vma->vm_flags & VM_LOCKED) {
@@ -1556,21 +1559,23 @@ static int acct_stack_growth(struct vm_a
    locked = mm->locked_vm + grow;
    limit = rlim[RLIMIT_MEMLOCK].rlim_cur >> PAGE_SHIFT;
    if (locked > limit && !capable(CAP_IPC_LOCK))
-    return -ENOMEM;
+    goto undo_charge;
   }

   /* Check to ensure the stack will not grow into a hugetlb-only region */
   new_start = (vma->vm_flags & VM_GROWSUP) ? vma->vm_start :
     vma->vm_end - size;
- if (is_hugepage_only_range(vma->vm_mm, new_start, size))
-  return -EFAULT;
+ if (is_hugepage_only_range(vma->vm_mm, new_start, size)) {
+  ret = -EFAULT;
+  goto undo_charge;
```

```
+ }

  /*
   * Overcommit..  This must be the final test, as it will
   * update security statistics.
   */
  if (security_vm_enough_memory(grow))
-  return -ENOMEM;
+  goto undo_charge;

  /* Ok, everything looks good - let it rip */
  mm->total_vm += grow;
@@ -1578,6 +1583,11 @@ static int acct_stack_growth(struct vm_a
   mm->locked_vm += grow;
  vm_stat_account(mm, vma->vm_flags, vma->vm_file, grow);
  return 0;
+undo_charge:
+ /* Undo memrlimit charge */
+ memrlimit_cgroup_uncharge_as(mm, grow);
+out:
+ return ret;
 }

 #if defined(CONFIG_STACK_GROWSUP) || defined(CONFIG_IA64)
@@ -1982,6 +1992,7 @@ unsigned long do_brk(unsigned long addr,
  struct rb_node ** rb_link, * rb_parent;
  pgoff_t pgoff = addr >> PAGE_SHIFT;
  int error;
+ int ret = -ENOMEM;

  len = PAGE_ALIGN(len);
  if (!len)
@@ -2035,13 +2046,13 @@ unsigned long do_brk(unsigned long addr,

  /* Check against address space limits *after* clearing old maps... */
  if (!may_expand_vm(mm, len >> PAGE_SHIFT))
-  return -ENOMEM;
+  return ret;

  if (mm->map_count > sysctl_max_map_count)
-  return -ENOMEM;
+  goto undo_charge;

  if (security_vm_enough_memory(len >> PAGE_SHIFT))
-  return -ENOMEM;
+  goto undo_charge;

  /* Can we just expand an old private anonymous mapping? */
```

```
  vma = vma_merge(mm, prev, addr, addr + len, flags,
@@ -2055,7 +2066,7 @@ unsigned long do_brk(unsigned long addr,
  vma = kmem_cache_zalloc(vm_area_cachep, GFP_KERNEL);
  if (!vma) {
   vm_unacct_memory(len >> PAGE_SHIFT);
-  return -ENOMEM;
+  goto undo_charge;
  }

  vma->vm_mm = mm;
@@ -2073,6 +2084,9 @@ out:
    mm->locked_vm += (len >> PAGE_SHIFT) - nr_pages;
  }
  return addr;
+undo_charge:
+ memrlimit_cgroup_uncharge_as(mm, len >> PAGE_SHIFT);
+ return ret;
 }

 EXPORT_SYMBOL(do_brk);
diff -puN mm/mremap.c~memrlimit-cgroup-add-better-error-handling mm/mremap.c
--- linux-2.6.26-rc5/mm/mremap.c~memrlimit-cgroup-add-better-error-handling 2008-06-19
21:12:46.000000000 +0530
+++ linux-2.6.26-rc5-balbir/mm/mremap.c 2008-06-19 22:00:02.000000000 +0530
@@ -18,6 +18,7 @@
 #include <linux/highmem.h>
 #include <linux/security.h>
 #include <linux/syscalls.h>
+#include <linux/memrlimitcgroup.h>

 #include <asm/uaccess.h>
 #include <asm/cacheflush.h>
@@ -256,6 +257,7 @@ unsigned long do_mremap(unsigned long ad
  struct vm_area_struct *vma;
  unsigned long ret = -EINVAL;
  unsigned long charged = 0;
+ int vm_expanded = 0;

  if (flags & ~(MREMAP_FIXED | MREMAP_MAYMOVE))
   goto out;
@@ -349,6 +351,7 @@ unsigned long do_mremap(unsigned long ad
   goto out;
  }

+ vm_expanded = 1;
  if (vma->vm_flags & VM_ACCOUNT) {
   charged = (new_len - old_len) >> PAGE_SHIFT;
   if (security_vm_enough_memory(charged))
```

```
@@ -411,6 +414,9 @@ out:
  if (ret & ~PAGE_MASK)
   vm_unacct_memory(charged);
 out_nc:
+ if (vm_expanded)
+  memrlimit_cgroup_uncharge_as(mm,
+    (new_len - old_len) >> PAGE_SHIFT);
  return ret;
 }


_
--
 Warm Regards,
 Balbir Singh
 Linux Technology Center
 IBM, ISTL
```

_____

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