Subject: [PATCH v2 11/29] cgroups: ability to stop res charge propagation on bounded ancestor
Posted by Glauber Costa on Fri, 11 May 2012 17:44:13 GMT
View Forum Message <> Reply to Message

From: Frederic Weisbecker <fweisbec@gmail.com>

Moving a task from a cgroup to another may require to substract its
resource charge from the old cgroup and add it to the new one.

For this to happen, the uncharge/charge propagation can just stop when we
reach the common ancestor for the two cgroups. Further the performance
reasons, we also want to avoid to temporarily overload the common
ancestors with a non-accurate resource counter usage if we charge first
the new cgroup and uncharge the old one thereafter. This is going to be a
requirement for the coming max number of task subsystem.

To solve this, provide a pair of new API that can charge/uncharge a
resource counter until we reach a given ancestor.

Signed-off-by: Frederic Weisbecker <fweisbec@gmail.com>
Acked-by: Paul Menage <paul@paulmenage.org>
Acked-by: Glauber Costa <glommer@parallels.com>
Cc: Li Zefan <lizf@cn.fujitsu.com>
Cc: Johannes Weiner <hannes@cmpxchg.org>
Cc: Aditya Kali <adityakali@google.com>
Cc: Oleg Nesterov <oleg@redhat.com>
Cc: Kay Sievers <kay.sievers@vrfy.org>
Cc: Tim Hockin <thockin@hockin.org>
Cc: Tejun Heo <htejun@gmail.com>
Acked-by: Kirill A. Shutemov <kirill@shutemov.name>
Signed-off-by: Andrew Morton <akpm@linux-foundation.org>

 Documentation/cgroups/resource_counter.txt | 18 ++++++++++++++++-
 include/linux/res_counter.h | 21 ++++++++++++++++++---
 kernel/res_counter.c | 13 ++++++++-----
 3 files changed, 43 insertions(+), 9 deletions(-)

diff --git a/Documentation/cgroups/resource_counter.txt
b/Documentation/cgroups/resource_counter.txt
index 95b24d7..a2cd05b 100644
--- a/Documentation/cgroups/resource_counter.txt
+++ b/Documentation/cgroups/resource_counter.txt
@@ -83,7 +83,15 @@ to work with it.
  res_counter->lock internally (it must be called with res_counter->lock
  held).

- e. void res_counter_uncharge[_locked]

```
+ e. int res_counter_charge_until(struct res_counter *counter,
+        struct res_counter *limit, unsigned long val,
+        struct res_counter **limit_fail_at)
+
+ The same as res_counter_charge(), but the charge propagation to
+ the hierarchy stops at the limit given in the "limit" parameter.
+
+
+ f. void res_counter_uncharge[_locked]
    (struct res_counter *rc, unsigned long val)

  When a resource is released (freed) it should be de-accounted
@@ -92,6 +100,14 @@ to work with it.

  The _locked routines imply that the res_counter->lock is taken.

+
+ g. void res_counter_uncharge_until(struct res_counter *counter,
+    struct res_counter *limit,
+    unsigned long val)
+
+ The same as res_counter_charge, but the uncharge propagation to
+ the hierarchy stops at the limit given in the "limit" parameter.
+
  2.1 Other accounting routines

     There are more routines that may help you with common needs, like
diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
index f7621cf..c12143e 100644
--- a/include/linux/res_counter.h
+++ b/include/linux/res_counter.h
@@ -117,11 +117,20 @@ void res_counter_init(struct res_counter *counter, struct res_counter
*parent);

 int __must_check res_counter_charge_locked(struct res_counter *counter,
   unsigned long val);
-int __must_check res_counter_charge(struct res_counter *counter,
-  unsigned long val, struct res_counter **limit_fail_at);
+int __must_check res_counter_charge_until(struct res_counter *counter,
+       struct res_counter *limit,
+       unsigned long val,
+       struct res_counter **limit_fail_at);
 int res_counter_charge_nofail(struct res_counter *counter,
   unsigned long val, struct res_counter **limit_fail_at);

+static inline int __must_check
+res_counter_charge(struct res_counter *counter, unsigned long val,
+     struct res_counter **limit_fail_at)
```

```
+{
+ return res_counter_charge_until(counter, NULL, val, limit_fail_at);
+}
+
 /*
  * uncharge - tell that some portion of the resource is released
  *
@@ -133,7 +142,13 @@ int res_counter_charge_nofail(struct res_counter *counter,
  */

 void res_counter_uncharge_locked(struct res_counter *counter, unsigned long val);
-void res_counter_uncharge(struct res_counter *counter, unsigned long val);
+void res_counter_uncharge_until(struct res_counter *counter,
+    struct res_counter *limit, unsigned long val);
+static inline void res_counter_uncharge(struct res_counter *counter,
+     unsigned long val)
+{
+ res_counter_uncharge_until(counter, NULL, val);
+}

 /**
  * res_counter_margin - calculate chargeable space of a counter
diff --git a/kernel/res_counter.c b/kernel/res_counter.c
index d508363..7aebf96 100644
--- a/kernel/res_counter.c
+++ b/kernel/res_counter.c
@@ -35,8 +35,9 @@ int res_counter_charge_locked(struct res_counter *counter, unsigned long
val)
  return 0;
 }

-int res_counter_charge(struct res_counter *counter, unsigned long val,
-   struct res_counter **limit_fail_at)
+int res_counter_charge_until(struct res_counter *counter,
+        struct res_counter *limit, unsigned long val,
+        struct res_counter **limit_fail_at)
 {
  int ret;
  unsigned long flags;
@@ -44,7 +45,7 @@ int res_counter_charge(struct res_counter *counter, unsigned long val,

  *limit_fail_at = NULL;
  local_irq_save(flags);
- for (c = counter; c != NULL; c = c->parent) {
+ for (c = counter; c != limit; c = c->parent) {
   spin_lock(&c->lock);
   ret = res_counter_charge_locked(c, val);
   spin_unlock(&c->lock);
```

```
@@ -99,13 +100,15 @@ void res_counter_uncharge_locked(struct res_counter *counter,
unsigned long val)
  counter->usage -= val;
 }

-void res_counter_uncharge(struct res_counter *counter, unsigned long val)
+void res_counter_uncharge_until(struct res_counter *counter,
+    struct res_counter *limit,
+    unsigned long val)
 {
  unsigned long flags;
  struct res_counter *c;

  local_irq_save(flags);
- for (c = counter; c != NULL; c = c->parent) {
+ for (c = counter; c != limit; c = c->parent) {
   spin_lock(&c->lock);
   res_counter_uncharge_locked(c, val);
   spin_unlock(&c->lock);
--
1.7.7.6
```