

This allows us two things basically:

1. If the subgroup has the limit higher than its parent has then the one will get more memory than allowed.
2. When we will need to account for a resource in more than one place, we'll be able to use this technics.

Look, consider we have a memory limit and swap limit. The memory limit is the limit for the sum of RSS, page cache and swap usage. To account for this gracefully, we'll set two counters:

```
res_counter mem_counter;  
res_counter swap_counter;
```

attach mm to the swap one

```
mm->mem_cnt = &swap_counter;
```

and make the swap_counter be mem's child. That's it. If we want hierarchical support, then the tree will look like this:

```
mem_counter_top  
swap_counter_top <- mm_struct living at top  
mem_counter_sub  
swap_counter_sub <- mm_struct living at sub
```

Signed-off-by: Pavel Emelianov <xemul@openvz.org>

```
include/linux/res_counter.h | 11 ++++++++-  
kernel/res_counter.c       | 36 ++++++++-----  
mm/memcontrol.c           | 9 +++++-  
3 files changed, 45 insertions(+), 11 deletions(-)
```

```
diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
```

```
index 2c4deb5..a27105e 100644
```

```
--- a/include/linux/res_counter.h
```

```
+++ b/include/linux/res_counter.h
```

```
@@ -41,6 +41,10 @@ struct res_counter {  
    * the routines below consider this to be IRQ-safe  
    */  
    spinlock_t lock;  
+ /*
```

```

+ * the parent counter. used for hierarchical resource accounting
+ */
+ struct res_counter *parent;
+ };

/**
@@ -80,7 +84,12 @@ enum {
+ * helpers for accounting
+ */

-void res_counter_init(struct res_counter *counter);
+/*
+ * the parent pointer is set only once - during the counter
+ * initialization. caller then must itself provide that this
+ * pointer is valid during the new counter lifetime
+ */
+void res_counter_init(struct res_counter *counter, struct res_counter *parent);

/*
+ * charge - try to consume more resource.
diff --git a/kernel/res_counter.c b/kernel/res_counter.c
index f1f20c2..046f6f4 100644
--- a/kernel/res_counter.c
+++ b/kernel/res_counter.c
@@ -13,10 +13,11 @@
#include <linux/res_counter.h>
#include <linux/uaccess.h>

-void res_counter_init(struct res_counter *counter)
+void res_counter_init(struct res_counter *counter, struct res_counter *parent)
{
+ spin_lock_init(&counter->lock);
+ counter->limit = (unsigned long long)LLONG_MAX;
+ counter->parent = parent;
}

int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
@@ -36,10 +37,26 @@ int res_counter_charge(struct res_counter *counter, unsigned long val)
{
+ int ret;
+ unsigned long flags;
+ struct res_counter *c, *unroll_c;
+
+ local_irq_save(flags);
+ for (c = counter; c != NULL; c = c->parent) {
+ spin_lock(&c->lock);
+ ret = res_counter_charge_locked(c, val);
+ spin_unlock(&c->lock);

```

```

+ if (ret < 0)
+ goto unroll;
+ }
+ local_irq_restore(flags);
+ return 0;

- spin_lock_irqsave(&counter->lock, flags);
- ret = res_counter_charge_locked(counter, val);
- spin_unlock_irqrestore(&counter->lock, flags);
+unroll:
+ for (unroll_c = counter; unroll_c != c; unroll_c = unroll_c->parent) {
+ spin_lock(&unroll_c->lock);
+ res_counter_uncharge_locked(unroll_c, val);
+ spin_unlock(&unroll_c->lock);
+ }
+ local_irq_restore(flags);
+ return ret;
}

```

```

@@ -54,10 +71,15 @@ void res_counter_uncharge_locked(struct res_counter *counter,
unsigned long val)

```

```

void res_counter_uncharge(struct res_counter *counter, unsigned long val)

```

```

{
+ unsigned long flags;
+ struct res_counter *c;

- spin_lock_irqsave(&counter->lock, flags);
- res_counter_uncharge_locked(counter, val);
- spin_unlock_irqrestore(&counter->lock, flags);
+ local_irq_save(flags);
+ for (c = counter; c != NULL; c = c->parent) {
+ spin_lock(&c->lock);
+ res_counter_uncharge_locked(c, val);
+ spin_unlock(&c->lock);
+ }
+ local_irq_restore(flags);
}

```

```

diff --git a/mm/memcontrol.c b/mm/memcontrol.c

```

```

index e5c741a..61db79c 100644

```

```

--- a/mm/memcontrol.c

```

```

+++ b/mm/memcontrol.c

```

```

@@ -976,19 +976,22 @@ static void free_mem_cgroup_per_zone_info(struct mem_cgroup
*mem, int node)

```

```

static struct cgroup_subsys_state *
mem_cgroup_create(struct cgroup_subsys *ss, struct cgroup *cont)
{

```

```

- struct mem_cgroup *mem;
+ struct mem_cgroup *mem, *parent;
  int node;

  if (unlikely((cont->parent) == NULL)) {
    mem = &init_mem_cgroup;
    init_mm.mem_cgroup = mem;
- } else
+ parent = NULL;
+ } else {
    mem = kzalloc(sizeof(struct mem_cgroup), GFP_KERNEL);
+ parent = mem_cgroup_from_cont(cont->parent);
+ }

  if (mem == NULL)
    return ERR_PTR(-ENOMEM);

- res_counter_init(&mem->res);
+ res_counter_init(&mem->res, parent ? &parent->res : NULL);

  memset(&mem->info, 0, sizeof(mem->info));

```

--
1.5.3.4

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