Subject: [RFC 3/7] bundle a percpu counter into res_counters and use its lock
Posted by Glauber Costa on Fri, 30 Mar 2012 08:04:41 GMT
View Forum Message <> Reply to Message

This is a preparation patch.
It bundles a percpu_counter into the resource counter. But it doesn't
do accounting with it just yet.

Instead. this preparation patch removes the res_counter spinlock,
and rely on the percpu_counter own lock for that.

Over time, this need to be done with acessors if we really plan to merge
it. But right now it can be used to give an idea about how it might be.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 include/linux/res_counter.h | 30 ++++++++++++++----------
 kernel/res_counter.c | 15 ++++++++-------
 2 files changed, 21 insertions(+), 24 deletions(-)

```
diff --git a/include/linux/res_counter.h b/include/linux/res_counter.h
index a860183..d4f3674 100644
--- a/include/linux/res_counter.h
+++ b/include/linux/res_counter.h
@@ -26,6 +26,7 @@ struct res_counter {
   * the current resource consumption level
   */
  unsigned long long usage;
+ struct percpu_counter usage_pcp;
  /*
   * the maximal value of the usage from the counter creation
   */
@@ -43,11 +44,6 @@ struct res_counter {
   */
  unsigned long long failcnt;
  /*
-  * the lock to protect all of the above.
-  * the routines below consider this to be IRQ-safe
-  */
- spinlock_t lock;
- /*
   * Parent counter, used for hierarchial resource accounting
   */
  struct res_counter *parent;
@@ -143,12 +139,12 @@ static inline unsigned long long res_counter_margin(struct res_counter
*cnt)
  unsigned long long margin;
  unsigned long flags;
```

```
- spin_lock_irqsave(&cnt->lock, flags);
+ raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
  if (cnt->limit > cnt->usage)
   margin = cnt->limit - cnt->usage;
  else
   margin = 0;
- spin_unlock_irqrestore(&cnt->lock, flags);
+ raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
  return margin;
 }

@@ -165,12 +161,12 @@ res_counter_soft_limit_excess(struct res_counter *cnt)
  unsigned long long excess;
  unsigned long flags;

- spin_lock_irqsave(&cnt->lock, flags);
+ raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
  if (cnt->usage <= cnt->soft_limit)
   excess = 0;
  else
   excess = cnt->usage - cnt->soft_limit;
- spin_unlock_irqrestore(&cnt->lock, flags);
+ raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
  return excess;
 }

@@ -178,18 +174,18 @@ static inline void res_counter_reset_max(struct res_counter *cnt)
 {
  unsigned long flags;

- spin_lock_irqsave(&cnt->lock, flags);
+ raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
  cnt->max_usage = cnt->usage;
- spin_unlock_irqrestore(&cnt->lock, flags);
+ raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
 }

 static inline void res_counter_reset_failcnt(struct res_counter *cnt)
 {
  unsigned long flags;

- spin_lock_irqsave(&cnt->lock, flags);
+ raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
  cnt->failcnt = 0;
- spin_unlock_irqrestore(&cnt->lock, flags);
+ raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
 }
```

```
 static inline int res_counter_set_limit(struct res_counter *cnt,
@@ -198,12 +194,12 @@ static inline int res_counter_set_limit(struct res_counter *cnt,
  unsigned long flags;
  int ret = -EBUSY;

- spin_lock_irqsave(&cnt->lock, flags);
+ raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
  if (cnt->usage <= limit) {
   cnt->limit = limit;
   ret = 0;
  }
- spin_unlock_irqrestore(&cnt->lock, flags);
+ raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
  return ret;
 }

@@ -213,9 +209,9 @@ res_counter_set_soft_limit(struct res_counter *cnt,
 {
  unsigned long flags;

- spin_lock_irqsave(&cnt->lock, flags);
+ raw_spin_lock_irqsave(&cnt->usage_pcp.lock, flags);
  cnt->soft_limit = soft_limit;
- spin_unlock_irqrestore(&cnt->lock, flags);
+ raw_spin_unlock_irqrestore(&cnt->usage_pcp.lock, flags);
  return 0;
 }

diff --git a/kernel/res_counter.c b/kernel/res_counter.c
index ecb4aad..70c46c9 100644
--- a/kernel/res_counter.c
+++ b/kernel/res_counter.c
@@ -11,15 +11,16 @@
 #include <linux/parser.h>
 #include <linux/fs.h>
 #include <linux/res_counter.h>
+#include <linux/percpu_counter.h>
 #include <linux/uaccess.h>
 #include <linux/mm.h>

 void res_counter_init(struct res_counter *counter, struct res_counter *parent)
 {
- spin_lock_init(&counter->lock);
  counter->limit = RESOURCE_MAX;
  counter->soft_limit = RESOURCE_MAX;
  counter->parent = parent;
+ percpu_counter_init(&counter->usage_pcp, 0);
```

```
 }

 int __res_counter_add(struct res_counter *c, long val, bool fail)
@@ -27,7 +28,7 @@ int __res_counter_add(struct res_counter *c, long val, bool fail)
  int ret = 0;
  u64 usage;

- spin_lock(&c->lock);
+ raw_spin_lock(&c->usage_pcp.lock);

  usage = c->usage;

@@ -45,7 +46,7 @@ int __res_counter_add(struct res_counter *c, long val, bool fail)
   c->max_usage = usage;

 out:
- spin_unlock(&c->lock);
+ raw_spin_unlock(&c->usage_pcp.lock);
  return ret;

 }
@@ -137,9 +138,9 @@ u64 res_counter_read_u64(struct res_counter *counter, int member)
  unsigned long flags;
  u64 ret;

- spin_lock_irqsave(&counter->lock, flags);
+ raw_spin_lock_irqsave(&counter->usage_pcp.lock, flags);
  ret = *res_counter_member(counter, member);
- spin_unlock_irqrestore(&counter->lock, flags);
+ raw_spin_unlock_irqrestore(&counter->usage_pcp.lock, flags);

  return ret;
 }
@@ -187,9 +188,9 @@ int res_counter_write(struct res_counter *counter, int member,
   if (*end != '\0')
    return -EINVAL;
  }
- spin_lock_irqsave(&counter->lock, flags);
+ raw_spin_lock_irqsave(&counter->usage_pcp.lock, flags);
  val = res_counter_member(counter, member);
  *val = tmp;
- spin_unlock_irqrestore(&counter->lock, flags);
+ raw_spin_unlock_irqrestore(&counter->usage_pcp.lock, flags);
  return 0;
 }
--
1.7.4.1
```