Subject: Re: [PATCH] memory.min_usage (seqlock for res_counter)
Posted by KAMEZAWA Hiroyuki on Tue, 04 Dec 2007 10:54:36 GMT
View Forum Message <> Reply to Message

This is seqlock version res_counter.
Maybe this this will reduce # of spin_lock.

Pavel-san, How about this ?

-Kame
==
resource counter's spinlock can be seqlock. (res_counter doesn't include
any pointers.)

And it will be good to avoid atomic ops if we can when we just checks
value.

Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>

```
 include/linux/res_counter.h |   24 +++++++----------------
 kernel/res_counter.c        |   14 +++++++-------
 2 files changed, 14 insertions(+), 24 deletions(-)

Index: linux-2.6.24-rc3-mm2/include/linux/res_counter.h
===================================================================
--- linux-2.6.24-rc3-mm2.orig/include/linux/res_counter.h
+++ linux-2.6.24-rc3-mm2/include/linux/res_counter.h
@@ -12,6 +12,7 @@
  */

 #include <linux/cgroup.h>
+#include <linux/seqlock.h>

 /*
  * The core object. the cgroup that wishes to account for some
@@ -36,7 +37,7 @@ struct res_counter {
   * the routines below consider this to be IRQ-safe
   */
- spinlock_t lock;
+ seqlock_t lock;
 };

 /*
@@ -101,27 +102,17 @@ int res_counter_charge(struct res_counte
```

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

-static inline bool res_counter_limit_check_locked(struct res_counter *cnt)
-{
- if (cnt->usage < cnt->limit)
-  return true;
-
- return false;
-}
-
-/*
- * Helper function to detect if the cgroup is within it's limit or
- * not. It's currently called from cgroup_rss_prepare()
- */
 static inline bool res_counter_check_under_limit(struct res_counter *cnt)
 {
  bool ret;
- unsigned long flags;
+ unsigned long seq;

- spin_lock_irqsave(&cnt->lock, flags);
- ret = res_counter_limit_check_locked(cnt);
- spin_unlock_irqrestore(&cnt->lock, flags);
+ do {
+  seq = read_seqbegin(&cnt->lock);
+  ret = (cnt->usage < cnt->limit);
+ } while (read_seqretry(&cnt->lock, seq));
  return ret;
 }

+
 #endif
Index: linux-2.6.24-rc3-mm2/kernel/res_counter.c
===================================================================
--- linux-2.6.24-rc3-mm2.orig/kernel/res_counter.c
+++ linux-2.6.24-rc3-mm2/kernel/res_counter.c
@@ -15,7 +15,7 @@

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

@@ -35,9 +35,9 @@ int res_counter_charge(struct res_counte
  int ret;
```

```
  unsigned long flags;

- spin_lock_irqsave(&counter->lock, flags);
+ write_seqlock_irqsave(&counter->lock, flags);
  ret = res_counter_charge_locked(counter, val);
- spin_unlock_irqrestore(&counter->lock, flags);
+ write_sequnlock_irqrestore(&counter->lock, flags);
  return ret;
 }

@@ -53,9 +53,9 @@ void res_counter_uncharge(struct res_cou
 {
  unsigned long flags;

- spin_lock_irqsave(&counter->lock, flags);
+ write_seqlock_irqsave(&counter->lock, flags);
  res_counter_uncharge_locked(counter, val);
- spin_unlock_irqrestore(&counter->lock, flags);
+ write_sequnlock_irqrestore(&counter->lock, flags);
 }


@@ -122,10 +122,10 @@ ssize_t res_counter_write(struct res_cou
   if (*end != '\0')
    goto out_free;
  }
- spin_lock_irqsave(&counter->lock, flags);
+ write_seqlock_irqsave(&counter->lock, flags);
  val = res_counter_member(counter, member);
  *val = tmp;
- spin_unlock_irqrestore(&counter->lock, flags);
+ write_sequnlock_irqrestore(&counter->lock, flags);
  ret = nbytes;
 out_free:
  kfree(buf);
```
