Subject: [RFC][for -mm] memory controller enhancements for reclaiming take2 [6/8] high_low watermark for res_
Posted by KAMEZAWA Hiroyuki on Mon, 03 Dec 2007 09:40:50 GMT
View Forum Message <> Reply to Message

This patch adds high/low watermark parameter to res_counter.
and check routine.

splitted out from YAMAMOTO's background page reclaim for memory cgroup set.

TODO?
 - if res_counter's user doesn't want high/low watermark, res_counter_write()
   should ignore low <= high <= limit limitation ?

Changelog
 * added param watermark_state this can ba read without lock lock.

Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>

 include/linux/res_counter.h | 28 ++++++++++++++++++++++++++++
 kernel/res_counter.c | 42 ++-
 2 files changed, 69 insertions(+), 1 deletion(-)

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
@@ -19,6 +19,12 @@
  * the helpers described beyond
  */

+enum watermark_state {
+ RES_WMARK_BELOW_LOW,
+ RES_WMARK_ABOVE_LOW,
+ RES_WMARK_ABOVE_HIGH,
+};
+
 struct res_counter {
  /*
   * the current resource consumption level
@@ -33,10 +39,17 @@ struct res_counter {
   */
  unsigned long long failcnt;
  /*
+ * Watermarks. Must keep low <= high <= limit.
+ */

```
+ unsigned long long high_watermark;
+ unsigned long long low_watermark;
+ /*
   * the lock to protect all of the above.
   * the routines below consider this to be IRQ-safe
   */
  spinlock_t lock;
+ /* can be read without lock */
+ enum watermark_state watermark_state;
 };

 /*
@@ -66,6 +79,8 @@ enum {
  RES_USAGE,
  RES_LIMIT,
  RES_FAILCNT,
+ RES_HWMARK,
+ RES_LWMARK,
 };

 /*
@@ -124,4 +139,17 @@ static inline bool res_counter_check_und
  return ret;
 }

+/*
+ * Helper function for implementing high/low watermark to resource controller.
+ */
+static inline bool res_counter_below_low_watermark(struct res_counter *cnt)
+{
+ return (cnt->watermark_state == RES_WMARK_BELOW_LOW);
+}
+
+static inline bool res_counter_above_high_watermark(struct res_counter *cnt)
+{
+ return (cnt->watermark_state == RES_WMARK_ABOVE_HIGH);
+}
+
 #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
@@ -17,6 +17,9 @@ void res_counter_init(struct res_counter
 {
  spin_lock_init(&counter->lock);
  counter->limit = (unsigned long long)LLONG_MAX;
+ counter->low_watermark = (unsigned long long)LLONG_MAX;
```

```
+ counter->high_watermark = (unsigned long long)LLONG_MAX;
+ counter->watermark_state = RES_WMARK_BELOW_LOW;
 }

 int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
@@ -27,6 +30,12 @@ int res_counter_charge_locked(struct res
  }

  counter->usage += val;
+
+ if (counter->usage > counter->high_watermark)
+  counter->watermark_state = RES_WMARK_ABOVE_HIGH;
+ else if (counter->usage > counter->low_watermark)
+  counter->watermark_state = RES_WMARK_ABOVE_LOW;
+
  return 0;
 }

@@ -47,6 +56,11 @@ void res_counter_uncharge_locked(struct
   val = counter->usage;

  counter->usage -= val;
+
+ if (counter->usage < counter->low_watermark)
+  counter->watermark_state = RES_WMARK_BELOW_LOW;
+ else if (counter->usage < counter->high_watermark)
+  counter->watermark_state = RES_WMARK_ABOVE_LOW;
 }

 void res_counter_uncharge(struct res_counter *counter, unsigned long val)
@@ -69,6 +83,10 @@ res_counter_member(struct res_counter *c
   return &counter->limit;
  case RES_FAILCNT:
   return &counter->failcnt;
+ case RES_HWMARK:
+  return &counter->high_watermark;
+ case RES_LWMARK:
+  return &counter->low_watermark;
  };

  BUG();
@@ -123,12 +141,34 @@ ssize_t res_counter_write(struct res_cou
    goto out_free;
  }
  spin_lock_irqsave(&counter->lock, flags);
+ /*
+  * check low <= high <= limit.
+  */
```

```
+ switch (member) {
+  case RES_LIMIT:
+   if (counter->high_watermark > tmp)
+    goto unlock_free;
+   break;
+  case RES_HWMARK:
+   if (tmp < counter->low_watermark ||
+       tmp > counter->limit)
+    goto unlock_free;
+   break;
+  case RES_LWMARK:
+   if (tmp > counter->high_watermark)
+    goto unlock_free;
+   break;
+  default:
+   break;
+ }
  val = res_counter_member(counter, member);
  *val = tmp;
- spin_unlock_irqrestore(&counter->lock, flags);
  ret = nbytes;
+unlock_free:
+ spin_unlock_irqrestore(&counter->lock, flags);
 out_free:
  kfree(buf);
 out:
  return ret;
 }
+
```