
Subject: Re: [RFC][only for review] memory controller background reclaim [3/5]
high/low watermark support in
Posted by [Pavel Emelianov](#) on Wed, 28 Nov 2007 11:12:53 GMT
[View Forum Message](#) <> [Reply to Message](#)

KAMEZAWA Hiroyuki wrote:

```
> This patch adds high/low watermark parameter to res_counter.
> splitted out from YAMAMOTO's background page reclaim for memory cgroup set.
>
> Changes:
> * added param watermark_state this allows status check without lock.
>
>
> Signed-off-by: KAMEZAWA Hiroyuki <kamezawa.hiroyu@jp.fujitsu.com>
> From: YAMAMOTO Takashi <yamamoto@valinux.co.jp>
>
> include/linux/res_counter.h | 27 ++++++
> kernel/res_counter.c        | 46 ++++++
> 2 files changed, 72 insertions(+), 1 deletion(-)
>
> Index: linux-2.6.24-rc3-mm1/include/linux/res_counter.h
> =====
> --- linux-2.6.24-rc3-mm1.orig/include/linux/res_counter.h 2007-11-28 14:33:19.000000000
+0900
> +++ linux-2.6.24-rc3-mm1/include/linux/res_counter.h 2007-11-28 16:37:32.000000000 +0900
> @@ -19,6 +19,12 @@
>  * the helpers described beyond
>  */
>
> +enum watermark_state {
> + RES_WATERMARK_BELOW_LOW,
> + RES_WATERMARK_ABOVE_LOW,
> + RES_WATERMARK_ABOVE_HIGH,
> +};
> +
> struct res_counter {
>  /*
>  * the current resource consumption level
>  @@ -33,10 +39,19 @@
>  */
>  unsigned long long failcnt;
>  /*
>  + * Watermarks
>  + * Should be changed automatically when the limit is changed and
>  + * 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;
> };
>
> /*
> @@ -73,6 +88,8 @@
>  RES_USAGE,
>  RES_LIMIT,
>  RES_FAILCNT,
> + RES_HIGH_WATERMARK,
> + RES_LOW_WATERMARK,

```

I'd prefer some shorter names. Like RES_HWMARK and RES_LWMARK.

```

> };
>
> /*
> @@ -131,4 +148,17 @@
>  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_WATERMARK_BELOW_LOW);
> +}
> +
> +static inline bool res_counter_above_high_watermark(struct res_counter *cnt)
> +{
> + return (cnt->watermark_state == RES_WATERMARK_ABOVE_HIGH);
> +}
> +
> #endif
> Index: linux-2.6.24-rc3-mm1/kernel/res_counter.c
> =====
> --- linux-2.6.24-rc3-mm1.orig/kernel/res_counter.c 2007-11-28 14:33:19.000000000 +0900
> +++ linux-2.6.24-rc3-mm1/kernel/res_counter.c 2007-11-28 16:33:20.000000000 +0900
> @@ -17,6 +17,9 @@
>  {
>  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_WATERMARK_BELOW_LOW;
> }
>
> int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
> @@ -27,6 +30,15 @@
> }
>
> counter->usage += val;
> +
> + if (counter->usage > counter->high_watermark) {
> + counter->watermark_state = RES_WATERMARK_ABOVE_HIGH;
> + return 0;
> + }

```

"else" would look much better here :)

```

> + if (counter->usage > counter->low_watermark)
> + counter->watermark_state = RES_WATERMARK_ABOVE_LOW;
> +
> return 0;
> }
>
> @@ -47,6 +59,13 @@
> val = counter->usage;
>
> counter->usage -= val;
> +
> + if (counter->usage < counter->low_watermark) {
> + counter->watermark_state = RES_WATERMARK_BELOW_LOW;
> + return;
> + }

```

and here

```

> + if (counter->usage < counter->high_watermark)
> + counter->watermark_state = RES_WATERMARK_ABOVE_LOW;
> }
>
> void res_counter_uncharge(struct res_counter *counter, unsigned long val)
> @@ -69,6 +88,10 @@
> return &counter->limit;
> case RES_FAILCNT:
> return &counter->failcnt;
> + case RES_HIGH_WATERMARK:
> + return &counter->high_watermark;
> + case RES_LOW_WATERMARK:

```

```

> + return &counter->low_watermark;
> };
>
> BUG();
> @@ -117,7 +140,7 @@
> {
> int ret;
> char *buf, *end;
> - unsigned long long flags, tmp, *val;
> + unsigned long long flags, tmp, *val, limit, low, high;
>
> buf = kmalloc(nbytes + 1, GFP_KERNEL);
> ret = -ENOMEM;
> @@ -141,6 +164,26 @@
> goto out_free;
> }
> spin_lock_irqsave(&counter->lock, flags);
> + /*
> + * High/Low watermark should be changed automatically AMAP.
> + */
> + switch (member) {
> + case RES_HIGH_WATERMARK:
> + limit = res_counter_get(counter, RES_LIMIT);
> + if (tmp > limit)
> + goto out_free;
> + low = res_counter_get(counter, RES_LOW_WATERMARK);
> + if (tmp <= low)
> + goto out_free;
> + break;
> + case RES_LOW_WATERMARK:
> + high = res_counter_get(counter, RES_HIGH_WATERMARK);
> + if (tmp >= high)
> + goto out_free;
> + break;

```

Why there's no checks for limit? Smth like

```

case RES_LIMIT:
    high = res_counter_get(counter, RES_HIGH_WATERMARK);
    if (tmp < high)
        goto out_free;

> + default:
> + break;
> + }
> val = res_counter_member(counter, member);
> *val = tmp;
> spin_unlock_irqrestore(&counter->lock, flags);

```

```
> @@ -150,3 +193,4 @@
> out:
>   return ret;
> }
> +
>
>
```

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