
Subject: [-mm PATCH] Memory controller improve user interface (v2)

Posted by [Balbir Singh](#) on Thu, 30 Aug 2007 18:52:46 GMT

[View Forum Message](#) <> [Reply to Message](#)

Change the interface to use kilobytes instead of pages. Page sizes can vary across platforms and configurations. A new strategy routine has been added to the resource counters infrastructure to format the data as desired.

Suggested by David Rientjes, Andrew Morton and Herbert Poetzl

Tested on a UML setup with the config for memory control enabled.

Signed-off-by: Balbir Singh <balbir@linux.vnet.ibm.com>

```
Documentation/controllers/memory.txt | 22 ++++++++
include/linux/res_counter.h          | 10 +++--
kernel/res_counter.c                 | 30 ++++++++-----
mm/memcontrol.c                     | 53 ++++++++-----
4 files changed, 90 insertions(+), 25 deletions(-)
```

```
diff -puN mm/memcontrol.c~mem-control-make-ui-more-usable mm/memcontrol.c
--- linux-2.6.23-rc3/mm/memcontrol.c~mem-control-make-ui-more-usable 2007-08-29
16:18:39.000000000 +0530
```

```
+++ linux-2.6.23-rc3-balbir/mm/memcontrol.c 2007-08-31 00:09:51.000000000 +0530
```

```
@@ -312,7 +312,7 @@ int mem_container_charge(struct page *pa
 * If we created the page_container, we should free it on exceeding
 * the container limit.
 */
```

```
- while (res_counter_charge(&mem->res, 1)) {
+ while (res_counter_charge(&mem->res, PAGE_SIZE)) {
    if (try_to_free_mem_container_pages(mem))
        continue;
```

```
@@ -352,7 +352,7 @@ int mem_container_charge(struct page *pa
    kfree(pc);
    pc = race_pc;
    atomic_inc(&pc->ref_cnt);
- res_counter_uncharge(&mem->res, 1);
+ res_counter_uncharge(&mem->res, PAGE_SIZE);
    css_put(&mem->css);
    goto done;
}
```

```
@@ -417,7 +417,7 @@ void mem_container_uncharge(struct page_
    css_put(&mem->css);
    page_assign_page_container(page, NULL);
    unlock_page_container(page);
- res_counter_uncharge(&mem->res, 1);
```

```

+ res_counter_uncharge(&mem->res, PAGE_SIZE);

    spin_lock_irqsave(&mem->lru_lock, flags);
    list_del_init(&pc->lru);
@@ -426,12 +426,44 @@ void mem_container_uncharge(struct page_
    }
}

-static ssize_t mem_container_read(struct container *cont, struct cftype *cft,
- struct file *file, char __user *userbuf, size_t nbytes,
- loff_t *ppos)
+/*
+ * Strategy routines for formatting read/write data
+ */
+int mem_container_read_strategy(unsigned long long val, char *buf)
+{
+ return sprintf(buf, "%llu Bytes\n", val);
+}
+
+int mem_container_write_strategy(char *buf, unsigned long long *tmp)
+{
+ *tmp = memparse(buf, &buf);
+ if (*buf != '\0')
+ return -EINVAL;
+
+ printk("tmp is %llu\n", *tmp);
+ /*
+ * Round up the value to the closest page size
+ */
+ *tmp = ((*tmp + PAGE_SIZE - 1) >> PAGE_SHIFT) << PAGE_SHIFT;
+ return 0;
+}
+
+static ssize_t mem_container_read_usage(struct container *cont,
+ struct cftype *cft, struct file *file,
+ char __user *userbuf, size_t nbytes, loff_t *ppos)
+{
+ return res_counter_read(&mem_container_from_cont(cont)->res,
+ cft->private, userbuf, nbytes, ppos,
+ mem_container_read_strategy);
+}
+
+static ssize_t mem_container_read(struct container *cont,
+ struct cftype *cft, struct file *file,
+ char __user *userbuf, size_t nbytes, loff_t *ppos)
+{
+ return res_counter_read(&mem_container_from_cont(cont)->res,
- cft->private, userbuf, nbytes, ppos);

```

```
+ cft->private, userbuf, nbytes, ppos,
+ NULL);
}
```

```
static ssize_t mem_container_write(struct container *cont, struct cftype *cft,
@@ -439,7 +471,8 @@ static ssize_t mem_container_write(struc
    size_t nbytes, loff_t *ppos)
{
    return res_counter_write(&mem_container_from_cont(cont)->res,
- cft->private, userbuf, nbytes, ppos);
+ cft->private, userbuf, nbytes, ppos,
+ mem_container_write_strategy);
}
```

```
static ssize_t mem_control_type_write(struct container *cont,
@@ -500,13 +533,13 @@ static struct cftype mem_container_files
{
    .name = "usage",
    .private = RES_USAGE,
- .read = mem_container_read,
+ .read = mem_container_read_usage,
},
{
    .name = "limit",
    .private = RES_LIMIT,
    .write = mem_container_write,
- .read = mem_container_read,
+ .read = mem_container_read_usage,
},
{
    .name = "failcnt",
```

```
diff -puN include/linux/memcontrol.h~mem-control-make-ui-more-usable
include/linux/memcontrol.h
```

```
diff -puN include/linux/res_counter.h~mem-control-make-ui-more-usable
include/linux/res_counter.h
```

```
--- linux-2.6.23-rc3/include/linux/res_counter.h~mem-control-make-ui-more-usable 2007-08-29
16:18:39.000000000 +0530
```

```
+++ linux-2.6.23-rc3-balbir/include/linux/res_counter.h 2007-08-30 23:48:49.000000000 +0530
```

```
@@ -23,11 +23,11 @@ struct res_counter {
```

```
/*
 * the current resource consumption level
 */
```

```
- unsigned long usage;
```

```
+ unsigned long long usage;
```

```
/*
 * the limit that usage cannot exceed
 */
```

```
- unsigned long limit;
```

```

+ unsigned long long limit;
/*
 * the number of unsuccessful attempts to consume the resource
 */
@@ -52,9 +52,11 @@ struct res_counter {
*/

ssize_t res_counter_read(struct res_counter *counter, int member,
- const char __user *buf, size_t nbytes, loff_t *pos);
+ const char __user *buf, size_t nbytes, loff_t *pos,
+ int (*read_strategy)(unsigned long long val, char *s));
ssize_t res_counter_write(struct res_counter *counter, int member,
- const char __user *buf, size_t nbytes, loff_t *pos);
+ const char __user *buf, size_t nbytes, loff_t *pos,
+ int (*write_strategy)(char *buf, unsigned long long *val));

/*
 * the field descriptors. one for each member of res_counter
diff -puN kernel/res_counter.c~mem-control-make-ui-more-usable kernel/res_counter.c
--- linux-2.6.23-rc3/kernel/res_counter.c~mem-control-make-ui-more-usable 2007-08-29
16:18:39.000000000 +0530
+++ linux-2.6.23-rc3-balbir/kernel/res_counter.c 2007-08-30 23:49:52.000000000 +0530
@@ -16,7 +16,7 @@
void res_counter_init(struct res_counter *counter)
{
    spin_lock_init(&counter->lock);
- counter->limit = (unsigned long)LONG_MAX;
+ counter->limit = (unsigned long long)LLONG_MAX;
}

int res_counter_charge_locked(struct res_counter *counter, unsigned long val)
@@ -76,24 +76,29 @@ static inline unsigned long *res_counter
}

ssize_t res_counter_read(struct res_counter *counter, int member,
- const char __user *userbuf, size_t nbytes, loff_t *pos)
+ const char __user *userbuf, size_t nbytes, loff_t *pos,
+ int (*read_strategy)(unsigned long long val, char *st_buf))
{
- unsigned long *val;
+ unsigned long long *val;
    char buf[64], *s;

    s = buf;
    val = res_counter_member(counter, member);
- s += sprintf(s, "%lu\n", *val);
+ if (read_strategy)
+ s += read_strategy(*val, s);

```

```

+ else
+ s += sprintf(s, "%lu\n", *val);
  return simple_read_from_buffer((void __user *)userbuf, nbytes,
    pos, buf, s - buf);
}

```

```

ssize_t res_counter_write(struct res_counter *counter, int member,
- const char __user *userbuf, size_t nbytes, loff_t *pos)
+ const char __user *userbuf, size_t nbytes, loff_t *pos,
+ int (*write_strategy)(char *st_buf, unsigned long long *val))
{
  int ret;
  char *buf, *end;
- unsigned long tmp, *val;
+ unsigned long long tmp, *val;

```

```

  buf = kmalloc(nbytes + 1, GFP_KERNEL);
  ret = -ENOMEM;
@@ -106,9 +111,16 @@ ssize_t res_counter_write(struct res_cou
  goto out_free;

```

```

  ret = -EINVAL;
- tmp = simple_strtoul(buf, &end, 10);
- if (*end != '\0')
- goto out_free;
+
+ if (write_strategy) {
+ if (write_strategy(buf, &tmp)) {
+ goto out_free;
+ }
+ } else {
+ tmp = simple_strtoul(buf, &end, 10);
+ if (*end != '\0')
+ goto out_free;
+ }

```

```

  val = res_counter_member(counter, member);
  *val = tmp;

```

```

diff -puN Documentation/controllers/memory.txt~mem-control-make-ui-more-usable
Documentation/controllers/memory.txt
---

```

```

linux-2.6.23-rc3/Documentation/controllers/memory.txt~mem-control-make-ui-more-usable 2007-0
8-29 16:18:39.000000000 +0530
+++ linux-2.6.23-rc3-balbir/Documentation/controllers/memory.txt 2007-08-31
00:19:31.000000000 +0530
@@ -165,11 +165,29 @@ c. Enable CONFIG_CONTAINER_MEM_CONT

```

Since now we're in the 0 container,

We can alter the memory limit:

```
-# echo -n 6000 > /containers/0/memory.limit
```

```
+# echo -n 4M > /containers/0/memory.limit
```

+

+NOTE: We can use a suffix (k, K, m, M, g or G) to indicate values in kilo, mega or gigabytes.

+

```
+# cat /containers/0/memory.limit
```

```
+4194304 Bytes
```

+

+NOTE: The interface has now changed to display the usage in bytes instead of pages

We can check the usage:

```
# cat /containers/0/memory.usage
```

```
-25
```

```
+1216512 Bytes
```

+

+Setting a limit to a number that is not a multiple of page size causes rounding up of the value. The user must check back to see (by reading memory.limit), to check for differences between desired values and committed values. Currently, all accounting is done in multiples of PAGE_SIZE

+

```
+# echo -n 1 > memory.limit
```

```
+# cat memory.limit
```

```
+4096 Bytes
```

The memory.failcnt field gives the number of times that the container limit was exceeded.

—

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Containers mailing list

Containers@lists.linux-foundation.org

<https://lists.linux-foundation.org/mailman/listinfo/containers>
