Subject: [-mm PATCH] Memory controller improve user interface (v3)
Posted by Balbir Singh on Sun, 02 Sep 2007 10:50:21 GMT
View Forum Message <> Reply to Message

Changelog for version 3
1. Change memory.limit and memory.usage to memory.limit_in_bytes and
   memory.usage_in_bytes respectively
2. Remove "Bytes" from the output of the limit and usage counters
3. Remove spurious printk

Changelog for version 2

1. Back end tracking is done in bytes, round up values of the limit if the
   specified value is not a multiple of page size. Display memory.usage
   and memory.limit in bytes (Dave Hansen, Paul Menage)

Change the interface to use bytes 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 | 28 +++++++++++++---
 include/linux/res_counter.h | 10 +++---
 kernel/res_counter.c | 30 +++++++++++-----
 mm/memcontrol.c | 56 +++++++++++++++++++++++++++++++--------
 4 files changed, 94 insertions(+), 30 deletions(-)

diff -puN Documentation/controllers/memory.txt~mem-control-make-ui-more-usable
Documentation/controllers/memory.txt
---
linux-2.6.23-rc4/Documentation/controllers/memory.txt~mem-control-make-ui-more-usable 2007-0
9-02 15:42:03.000000000 +0530
+++ linux-2.6.23-rc4-balbir/Documentation/controllers/memory.txt 2007-09-02
15:47:12.000000000 +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_in_bytes

```
+
+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_in_bytes
+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
+# cat /containers/0/memory.usage_in_bytes
+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_in_bytes), 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_in_bytes
+# cat memory.limit_in_bytes
+4096 Bytes

 The memory.failcnt field gives the number of times that the container limit was
 exceeded.
@@ -206,8 +224,8 @@ container might have some charge associa
 tasks have migrated away from it. If some pages are still left, after following
 the steps listed in sections 4.1 and 4.2, check the Swap Cache usage in
 /proc/meminfo to see if the Swap Cache usage is showing up in the
-containers memory.usage counter. A simple test of swapoff -a and swapon -a
-should free any pending Swap Cache usage.
+containers memory.usage_in_bytes counter. A simple test of swapoff -a and
+swapon -a should free any pending Swap Cache usage.

 4.4 Choosing what to account  -- Page Cache (unmapped) vs RSS (mapped)?

diff -puN include/linux/res_counter.h~mem-control-make-ui-more-usable
include/linux/res_counter.h
--- linux-2.6.23-rc4/include/linux/res_counter.h~mem-control-make-ui-more-usable 2007-09-02
15:42:03.000000000 +0530
+++ linux-2.6.23-rc4-balbir/include/linux/res_counter.h 2007-09-02 15:42:03.000000000 +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-rc4/kernel/res_counter.c~mem-control-make-ui-more-usable 2007-09-02
15:42:03.000000000 +0530
+++ linux-2.6.23-rc4-balbir/kernel/res_counter.c 2007-09-02 15:42:03.000000000 +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 mm/memcontrol.c~mem-control-make-ui-more-usable mm/memcontrol.c
--- linux-2.6.23-rc4/mm/memcontrol.c~mem-control-make-ui-more-usable 2007-09-02
15:42:03.000000000 +0530
+++ linux-2.6.23-rc4-balbir/mm/memcontrol.c 2007-09-02 15:45:49.000000000 +0530
@@ -313,7 +313,7 @@ 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;

@@ -353,7 +353,7 @@ 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;
  }
@@ -418,7 +418,7 @@ 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);
@@ -427,12 +427,43 @@ 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 formating read/write data
+ */
+int mem_container_read_strategy(unsigned long long val, char *buf)
+{
+ return sprintf(buf, "%llu\n", val);
+}
+
+int mem_container_write_strategy(char *buf, unsigned long long *tmp)
+{
+ *tmp = memparse(buf, &buf);
+ if (*buf != '\0')
+  return -EINVAL;
+
+ /*
```

```
+  * 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,
@@ -440,7 +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,
@@ -499,15 +531,15 @@ static ssize_t mem_control_type_read(str

 static struct cftype mem_container_files[] = {
   {
-   .name = "usage",
+   .name = "usage_in_bytes",
    .private = RES_USAGE,
-   .read = mem_container_read,
+   .read = mem_container_read_usage,
   },
   {
-   .name = "limit",
```

```
+  .name = "limit_in_bytes",
   .private = RES_LIMIT,
   .write = mem_container_write,
-  .read = mem_container_read,
+  .read = mem_container_read_usage,
  },
  {
   .name = "failcnt",
_
```

--
Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL