Subject: [PATCH 32/33] memory-controller-improve-user-interface
Posted by Paul Menage on Mon, 17 Sep 2007 21:03:39 GMT
View Forum Message <> Reply to Message

From: Balbir Singh <balbir@linux.vnet.ibm.com>
(container->cgroup renaming by Paul Menage <menage@google.com>)

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>
Signed-off-by: Pavel Emelianov <xemul@openvz.org>
Signed-off-by: Paul Menage <menage@google.com>

```
 Documentation/controllers/memory.txt |   29 ++++++++++++++++++---
 include/linux/res_counter.h          |   12 +++++---
 kernel/res_counter.c                 |   34 +++++++++++++++++++--------
 mm/memcontrol.c                      |   35 +++++++++++++++++++--------
 4 files changed, 79 insertions(+), 31 deletions(-)

diff -puN Documentation/controllers/memory.txt~memory-controller-improve-user-interface
Documentation/controllers/memory.txt
--- a/Documentation/controllers/memory.txt~memory-controller-improve-user-interface
+++ a/Documentation/controllers/memory.txt
@@ -165,11 +165,30 @@ c. Enable CONFIG_CGROUP_MEM_CONT

 Since now we're in the 0 cgroup,
 We can alter the memory limit:
-# echo -n 6000 > /cgroups/0/memory.limit
+# echo -n 4M > /cgroups/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 /cgroups/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 /cgroups/0/memory.usage
-25
+# cat /cgroups/0/memory.usage_in_bytes
+1216512 Bytes
+
+A successful write to this file does not guarantee a successful set of
+this limit to the value written into the file.  This can be due to a
+number of factors, such as rounding up to page boundaries or the total
+availability of memory on the system.  The user is required to re-read
+this file after a write to guarantee the value committed by the kernel.
+
+# 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 cgroup limit was
 exceeded.
@@ -206,8 +225,8 @@ cgroup 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
-cgroups memory.usage counter. A simple test of swapoff -a and swapon -a
-should free any pending Swap Cache usage.
+cgroups 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~memory-controller-improve-user-interface
include/linux/res_counter.h
--- a/include/linux/res_counter.h~memory-controller-improve-user-interface
+++ a/include/linux/res_counter.h
@@ -23,15 +23,15 @@ 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
   */
- unsigned long failcnt;
+ unsigned long long failcnt;
```

```
  /*
   * the lock to protect all of the above.
   * the routines below consider this to be IRQ-safe
@@ -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~memory-controller-improve-user-interface kernel/res_counter.c
--- a/kernel/res_counter.c~memory-controller-improve-user-interface
+++ a/kernel/res_counter.c
@@ -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)
@@ -59,8 +59,8 @@ void res_counter_uncharge(struct res_cou
 }


-static inline unsigned long *res_counter_member(struct res_counter *counter,
-      int member)
+static inline unsigned long long *
+res_counter_member(struct res_counter *counter, int member)
 {
  switch (member) {
  case RES_USAGE:
@@ -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, "%llu\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_strtoull(buf, &end, 10);
+  if (*end != '\0')
+   goto out_free;
+ }

  val = res_counter_member(counter, member);
  *val = tmp;
diff -puN mm/memcontrol.c~memory-controller-improve-user-interface mm/memcontrol.c
```

```
--- a/mm/memcontrol.c~memory-controller-improve-user-interface
+++ a/mm/memcontrol.c
@@ -296,7 +296,7 @@ int mem_cgroup_charge(struct page *pa
   * If we created the page_cgroup, we should free it on exceeding
   * the cgroup limit.
   */
- while (res_counter_charge(&mem->res, 1)) {
+ while (res_counter_charge(&mem->res, PAGE_SIZE)) {
   if (try_to_free_mem_cgroup_pages(mem))
    continue;

@@ -335,7 +335,7 @@ int mem_cgroup_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;
  }
@@ -380,7 +380,7 @@ void mem_cgroup_uncharge(struct page_
   css_put(&mem->css);
   page_assign_page_cgroup(page, NULL);
   unlock_page_cgroup(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);
@@ -389,12 +389,26 @@ void mem_cgroup_uncharge(struct page_
  }
 }

-static ssize_t mem_cgroup_read(struct cgroup *cont, struct cftype *cft,
-   struct file *file, char __user *userbuf, size_t nbytes,
-   loff_t *ppos)
+int mem_cgroup_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_cgroup_read(struct cgroup *cont,
+   struct cftype *cft, struct file *file,
+   char __user *userbuf, size_t nbytes, loff_t *ppos)
 {
  return res_counter_read(&mem_cgroup_from_cont(cont)->res,
-    cft->private, userbuf, nbytes, ppos);
+    cft->private, userbuf, nbytes, ppos,
+    NULL);
 }

 static ssize_t mem_cgroup_write(struct cgroup *cont, struct cftype *cft,
@@ -402,5 +416,6 @@ static ssize_t mem_cgroup_write(struc
     size_t nbytes, loff_t *ppos)
 {
  return res_counter_write(&mem_cgroup_from_cont(cont)->res,
-    cft->private, userbuf, nbytes, ppos);
+    cft->private, userbuf, nbytes, ppos,
+    mem_cgroup_write_strategy);
 }
@@ -407,12 +422,12 @@

 static struct cftype mem_cgroup_files[] = {
  {
-  .name = "usage",
+  .name = "usage_in_bytes",
   .private = RES_USAGE,
   .read = mem_cgroup_read,
  },
  {
-  .name = "limit",
+  .name = "limit_in_bytes",
   .private = RES_LIMIT,
   .write = mem_cgroup_write,
   .read = mem_cgroup_read,

--
```

_____

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