
Subject: [PATCH 6/8] CGroup Files: Remove cpuset_common_file_write()
Posted by [Paul Menage](#) on Fri, 20 Jun 2008 23:44:04 GMT
[View Forum Message](#) <> [Reply to Message](#)

This patch tweaks the signatures of the update_cpumask() and update_nodemask() functions so that they can be called directly as handlers for the new cgroups write_string() method.

This allows cpuset_common_file_write() to be removed.

Signed-off-by: Paul Menage <menage@google.com>

This patch fails checkpatch.pl which objects to the use of NR_CPUS. It's not clear that there's a way to express this more cleanly, other than leaving it empty and initializing it early in the boot process.

kernel/cpuset.c | 109 ++++++-----
1 file changed, 35 insertions(+), 74 deletions(-)

Index: cws-2.6.26-rc5-mm3/kernel/cpuset.c

=====

--- cws-2.6.26-rc5-mm3.orig/kernel/cpuset.c

+++ cws-2.6.26-rc5-mm3/kernel/cpuset.c

```
@@ -227,10 +227,6 @@ static struct cpuset top_cpuset = {
 * The task_struct fields mems_allowed and mems_generation may only
 * be accessed in the context of that task, so require no locks.
 *
- * The cpuset_common_file_write handler for operations that modify
- * the cpuset hierarchy holds cgroup_mutex across the entire operation,
- * single threading all such cpuset modifications across the system.
- *
 * The cpuset_common_file_read() handlers only hold callback_mutex across
 * small pieces of code, such as when reading out possibly multi-word
 * cpumasks and nodemasks.
@@ -801,7 +797,7 @@ static int update_tasks_cpumask(struct c
 * @cs: the cpuset to consider
 * @buf: buffer of cpu numbers written to this cpuset
 */
-static int update_cpumask(struct cpuset *cs, char *buf)
+static int update_cpumask(struct cpuset *cs, const char *buf)
{
    struct cpuset trialcs;
    int retval;
@@ -819,7 +815,6 @@ static int update_cpumask(struct cpuset
 * that parsing. The validate_change() call ensures that cpusets
```

```

    * with tasks have cpus.
    */
- buf = strstr(buf);
  if (!*buf) {
    cpus_clear(trialcs.cpus_allowed);
  } else {
@@ -1016,7 +1011,7 @@ done:
    * lock each such tasks mm->mmap_sem, scan its vma's and rebind
    * their mempolicies to the cpusets new mems_allowed.
    */
-static int update_nodemask(struct cpuset *cs, char *buf)
+static int update_nodemask(struct cpuset *cs, const char *buf)
{
  struct cpuset trialcs;
  nodemask_t oldmem;
@@ -1037,7 +1032,6 @@ static int update_nodemask(struct cpuset
    * that parsing. The validate_change() call ensures that cpusets
    * with tasks have memory.
    */
- buf = strstr(buf);
  if (!*buf) {
    nodes_clear(trialcs.mems_allowed);
  } else {
@@ -1280,72 +1274,14 @@ typedef enum {
  FILE_SPREAD_SLAB,
} cpuset_filetype_t;

-static ssize_t cpuset_common_file_write(struct cgroup *cont,
-    struct cftype *cft,
-    struct file *file,
-    const char __user *userbuf,
-    size_t nbytes, loff_t *unused_ppos)
-{
- struct cpuset *cs = cgroup_cs(cont);
- cpuset_filetype_t type = cft->private;
- char *buffer;
- int retval = 0;
-
- /* Crude upper limit on largest legitimate cpulist user might write. */
- if (nbytes > 100U + 6 * max(NR_CPUS, MAX_NUMNODES))
- return -E2BIG;
-
- /* +1 for nul-terminator */
- buffer = kmalloc(nbytes + 1, GFP_KERNEL);
- if (!buffer)
- return -ENOMEM;
-
- if (copy_from_user(buffer, userbuf, nbytes)) {

```

```

- retval = -EFAULT;
- goto out1;
- }
- buffer[nbytes] = 0; /* nul-terminate */
-
- cgroup_lock();
-
- if (cgroup_is_removed(cont)) {
- retval = -ENODEV;
- goto out2;
- }
-
- switch (type) {
- case FILE_CPULIST:
- retval = update_cpumask(cs, buffer);
- break;
- case FILE_MEMLIST:
- retval = update_nodemask(cs, buffer);
- break;
- default:
- retval = -EINVAL;
- goto out2;
- }
-
- if (retval == 0)
- retval = nbytes;
-out2:
- cgroup_unlock();
-out1:
- kfree(buffer);
- return retval;
-}
-
static int cpuset_write_u64(struct cgroup *cgrp, struct cftype *cft, u64 val)
{
    int retval = 0;
    struct cpuset *cs = cgroup_cs(cgrp);
    cpuset_filetype_t type = cft->private;

- cgroup_lock();
-
- if (cgroup_is_removed(cgrp)) {
- cgroup_unlock();
+ if (!cgroup_lock_live_group(cgrp))
    return -ENODEV;
- }

    switch (type) {

```

```

    case FILE_CPU_EXCLUSIVE:
@@ -1391,12 +1327,9 @@ static int cpuset_write_s64(struct cgrou
    struct cpuset *cs = cgroup_cs(cgrp);
    cpuset_filetype_t type = cft->private;

- cgroup_lock();
-
- if (cgroup_is_removed(cgrp)) {
- cgroup_unlock();
+ if (!cgroup_lock_live_group(cgrp))
    return -ENODEV;
- }
+
    switch (type) {
    case FILE_SCHED_RELAX_DOMAIN_LEVEL:
        retval = update_relax_domain_level(cs, val);
@@ -1410,6 +1343,32 @@ static int cpuset_write_s64(struct cgrou
    }

/*
+ * Common handling for a write to a "cpus" or "mems" file.
+ */
+static int cpuset_write_resmask(struct cgroup *cgrp, struct cftype *cft,
+    const char *buf)
+{
+    int retval = 0;
+
+    if (!cgroup_lock_live_group(cgrp))
+        return -ENODEV;
+
+    switch (cft->private) {
+    case FILE_CPULIST:
+        retval = update_cpumask(cgroup_cs(cgrp), buf);
+        break;
+    case FILE_MEMLIST:
+        retval = update_nodemask(cgroup_cs(cgrp), buf);
+        break;
+    default:
+        retval = -EINVAL;
+        break;
+    }
+    cgroup_unlock();
+    return retval;
+}
+
+/*
+ * These ascii lists should be read in a single call, by using a user
+ * buffer large enough to hold the entire map. If read in smaller

```

* chunks, there is no guarantee of atomicity. Since the display format

@@ -1528,14 +1487,16 @@ static struct cftype files[] = {

```
{
    .name = "cpus",
    .read = cpuset_common_file_read,
-   .write = cpuset_common_file_write,
+   .write_string = cpuset_write_resmask,
+   .max_write_len = (100U + 6 * NR_CPUS),
    .private = FILE_CPULIST,
},

{
    .name = "mems",
    .read = cpuset_common_file_read,
-   .write = cpuset_common_file_write,
+   .write_string = cpuset_write_resmask,
+   .max_write_len = (100U + 6 * MAX_NUMNODES),
    .private = FILE_MEMLIST,
},
```

--

Containers mailing list

Containers@lists.linux-foundation.org

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