Subject: [PATCH v2 3/5] change number_of_cpusets to an atomic
Posted by Glauber Costa on Mon, 23 Apr 2012 19:37:45 GMT
View Forum Message <> Reply to Message

This will allow us to call destroy() without holding the
cgroup_mutex(). Other important updates inside update_flags()
are protected by the callback_mutex.

We could protect this variable with the callback_mutex as well,
as suggested by Li Zefan, but we need to make sure we are protected
by that mutex at all times, and some of its updates happen inside the
cgroup_mutex - which means we would deadlock.

An atomic variable is not expensive, since it is seldom updated,
and protect us well.

Signed-off-by: Glauber Costa <glommer@parallels.com>
---
 include/linux/cpuset.h | 6 +++---
 kernel/cpuset.c | 10 +++++++-----
 2 files changed, 8 insertions(+), 8 deletions(-)

diff --git a/include/linux/cpuset.h b/include/linux/cpuset.h
index 668f66b..9b3d468 100644
--- a/include/linux/cpuset.h
+++ b/include/linux/cpuset.h
@@ -16,7 +16,7 @@

 #ifdef CONFIG_CPUSETS

-extern int number_of_cpusets; /* How many cpusets are defined in system? */
+extern atomic_t number_of_cpusets; /* How many cpusets are defined in system? */

 extern int cpuset_init(void);
 extern void cpuset_init_smp(void);
@@ -33,13 +33,13 @@ extern int __cpuset_node_allowed_hardwall(int node, gfp_t gfp_mask);

 static inline int cpuset_node_allowed_softwall(int node, gfp_t gfp_mask)
 {
- return number_of_cpusets <= 1 ||
+ return atomic_read(&number_of_cpusets) <= 1 ||
    __cpuset_node_allowed_softwall(node, gfp_mask);
 }

 static inline int cpuset_node_allowed_hardwall(int node, gfp_t gfp_mask)
 {
- return number_of_cpusets <= 1 ||
+ return atomic_read(&number_of_cpusets) <= 1 ||

```
   __cpuset_node_allowed_hardwall(node, gfp_mask);
 }

diff --git a/kernel/cpuset.c b/kernel/cpuset.c
index 8c8bd65..65bfd6d 100644
--- a/kernel/cpuset.c
+++ b/kernel/cpuset.c
@@ -73,7 +73,7 @@ static struct workqueue_struct *cpuset_wq;
  * When there is only one cpuset (the root cpuset) we can
  * short circuit some hooks.
  */
-int number_of_cpusets __read_mostly;
+atomic_t number_of_cpusets __read_mostly;

 /* Forward declare cgroup structures */
 struct cgroup_subsys cpuset_subsys;
@@ -583,7 +583,7 @@ static int generate_sched_domains(cpumask_var_t **domains,
   goto done;
  }

- csa = kmalloc(number_of_cpusets * sizeof(cp), GFP_KERNEL);
+ csa = kmalloc(atomic_read(&number_of_cpusets) * sizeof(cp), GFP_KERNEL);
  if (!csa)
   goto done;
  csn = 0;
@@ -1848,7 +1848,7 @@ static struct cgroup_subsys_state *cpuset_create(struct cgroup *cont)
  cs->relax_domain_level = -1;

  cs->parent = parent;
- number_of_cpusets++;
+ atomic_inc(&number_of_cpusets);
  return &cs->css ;
 }

@@ -1865,7 +1865,7 @@ static void cpuset_destroy(struct cgroup *cont)
  if (is_sched_load_balance(cs))
   update_flag(CS_SCHED_LOAD_BALANCE, cs, 0);

- number_of_cpusets--;
+ atomic_dec(&number_of_cpusets);
  free_cpumask_var(cs->cpus_allowed);
  kfree(cs);
 }
@@ -1909,7 +1909,7 @@ int __init cpuset_init(void)
  if (!alloc_cpumask_var(&cpus_attach, GFP_KERNEL))
   BUG();

- number_of_cpusets = 1;
```

```
+ atomic_set(&number_of_cpusets, 1);
  return 0;
 }

--
1.7.7.6
```