Subject: [PATCH v8 2/3] cgroups: add per-thread subsystem callbacks
Posted by Ben Blum on Tue, 08 Feb 2011 01:39:15 GMT
View Forum Message <> Reply to Message

Add cgroup subsystem callbacks for per-thread attachment

From: Ben Blum <bblum@andrew.cmu.edu>

This patch adds can_attach_task, pre_attach, and attach_task as new callbacks for cgroups's subsystem interface. Unlike can_attach and attach, these are for per-thread operations, to be called potentially many times when attaching an entire threadgroup.

Also, the old "bool threadgroup" interface is removed, as replaced by this. All subsystems are modified for the new interface - of note is cpuset, which requires from/to nodemasks for attach to be globally scoped (though per-cpuset would work too) to persist from its pre_attach to attach_task and attach.

This is a pre-patch for cgroup-procs-writable.patch.

Signed-off-by: Ben Blum <bblum@andrew.cmu.edu>

```
---
 Documentation/cgroups/cgroups.txt |   30 ++++++++---
 block/blk-cgroup.c                |   18 ++----
 include/linux/cgroup.h            |   10 ++--
 kernel/cgroup.c                   |   17 +++++-
 kernel/cgroup_freezer.c           |   26 ++++-----
 kernel/cpuset.c                   |  105 ++++++++++++++++--------------------
 kernel/ns_cgroup.c                |   23 +++-----
 kernel/sched.c                    |   38 +------------
 mm/memcontrol.c                   |   18 ++----
 security/device_cgroup.c          |    3 -
 10 files changed, 122 insertions(+), 166 deletions(-)

diff --git a/Documentation/cgroups/cgroups.txt b/Documentation/cgroups/cgroups.txt
index 190018b..d3c9a24 100644
--- a/Documentation/cgroups/cgroups.txt
+++ b/Documentation/cgroups/cgroups.txt
@@ -563,7 +563,7 @@ rmdir() will fail with it. From this behavior, pre_destroy() can be
 called multiple times against a cgroup.

 int can_attach(struct cgroup_subsys *ss, struct cgroup *cgrp,
-	       struct task_struct *task, bool threadgroup)
+	       struct task_struct *task)
 (cgroup_mutex held by caller)

 Called prior to moving a task into a cgroup; if the subsystem
@@ -572,9 +572,14 @@ task is passed, then a successful result indicates that *any*
```

```
 unspecified task can be moved into the cgroup. Note that this isn't
 called on a fork. If this method returns 0 (success) then this should
 remain valid while the caller holds cgroup_mutex and it is ensured that either
-attach() or cancel_attach() will be called in future. If threadgroup is
-true, then a successful result indicates that all threads in the given
-thread's threadgroup can be moved together.
+attach() or cancel_attach() will be called in future.
+
+int can_attach_task(struct cgroup *cgrp, struct task_struct *tsk);
+(cgroup_mutex held by caller)
+
+As can_attach, but for operations that must be run once per task to be
+attached (possibly many when using cgroup_attach_proc). Called after
+can_attach.

 void cancel_attach(struct cgroup_subsys *ss, struct cgroup *cgrp,
        struct task_struct *task, bool threadgroup)
@@ -586,15 +591,24 @@ function, so that the subsystem can implement a rollback. If not, not
necessary.
 This will be called only about subsystems whose can_attach() operation have
 succeeded.

+void pre_attach(struct cgroup *cgrp);
+(cgroup_mutex held by caller)
+
+For any non-per-thread attachment work that needs to happen before
+attach_task. Needed by cpuset.
+
 void attach(struct cgroup_subsys *ss, struct cgroup *cgrp,
-     struct cgroup *old_cgrp, struct task_struct *task,
-     bool threadgroup)
+     struct cgroup *old_cgrp, struct task_struct *task)
 (cgroup_mutex held by caller)

 Called after the task has been attached to the cgroup, to allow any
 post-attachment activity that requires memory allocations or blocking.
-If threadgroup is true, the subsystem should take care of all threads
-in the specified thread's threadgroup. Currently does not support any
+
+void attach_task(struct cgroup *cgrp, struct task_struct *tsk);
+(cgroup_mutex held by caller)
+
+As attach, but for operations that must be run once per task to be attached,
+like can_attach_task. Called before attach. Currently does not support any
 subsystem that might need the old_cgrp for every thread in the group.

 void fork(struct cgroup_subsy *ss, struct task_struct *task)
diff --git a/block/blk-cgroup.c b/block/blk-cgroup.c
```

```
index b1febd0..45b3809 100644
--- a/block/blk-cgroup.c
+++ b/block/blk-cgroup.c
@@ -30,10 +30,8 @@ EXPORT_SYMBOL_GPL(blkio_root_cgroup);

 static struct cgroup_subsys_state *blkiocg_create(struct cgroup_subsys *,
         struct cgroup *);
-static int blkiocg_can_attach(struct cgroup_subsys *, struct cgroup *,
-         struct task_struct *, bool);
-static void blkiocg_attach(struct cgroup_subsys *, struct cgroup *,
-       struct cgroup *, struct task_struct *, bool);
+static int blkiocg_can_attach_task(struct cgroup *, struct task_struct *);
+static void blkiocg_attach_task(struct cgroup *, struct task_struct *);
 static void blkiocg_destroy(struct cgroup_subsys *, struct cgroup *);
 static int blkiocg_populate(struct cgroup_subsys *, struct cgroup *);

@@ -46,8 +44,8 @@ static int blkiocg_populate(struct cgroup_subsys *, struct cgroup *);
 struct cgroup_subsys blkio_subsys = {
   .name = "blkio",
   .create = blkiocg_create,
- .can_attach = blkiocg_can_attach,
- .attach = blkiocg_attach,
+ .can_attach_task = blkiocg_can_attach_task,
+ .attach_task = blkiocg_attach_task,
   .destroy = blkiocg_destroy,
   .populate = blkiocg_populate,
 #ifdef CONFIG_BLK_CGROUP
@@ -1475,9 +1473,7 @@ done:
  * of the main cic data structures.  For now we allow a task to change
  * its cgroup only if it's the only owner of its ioc.
  */
-static int blkiocg_can_attach(struct cgroup_subsys *subsys,
-    struct cgroup *cgroup, struct task_struct *tsk,
-    bool threadgroup)
+static int blkiocg_can_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
 {
   struct io_context *ioc;
   int ret = 0;
@@ -1492,9 +1488,7 @@ static int blkiocg_can_attach(struct cgroup_subsys *subsys,
   return ret;
 }

-static void blkiocg_attach(struct cgroup_subsys *subsys, struct cgroup *cgroup,
-    struct cgroup *prev, struct task_struct *tsk,
-    bool threadgroup)
+static void blkiocg_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
 {
   struct io_context *ioc;
```

```
diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index ce104e3..35b69b4 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -467,12 +467,14 @@ struct cgroup_subsys {
  int (*pre_destroy)(struct cgroup_subsys *ss, struct cgroup *cgrp);
  void (*destroy)(struct cgroup_subsys *ss, struct cgroup *cgrp);
  int (*can_attach)(struct cgroup_subsys *ss, struct cgroup *cgrp,
-     struct task_struct *tsk, bool threadgroup);
+     struct task_struct *tsk);
+ int (*can_attach_task)(struct cgroup *cgrp, struct task_struct *tsk);
  void (*cancel_attach)(struct cgroup_subsys *ss, struct cgroup *cgrp,
-     struct task_struct *tsk, bool threadgroup);
+         struct task_struct *tsk);
+ void (*pre_attach)(struct cgroup *cgrp);
+ void (*attach_task)(struct cgroup *cgrp, struct task_struct *tsk);
  void (*attach)(struct cgroup_subsys *ss, struct cgroup *cgrp,
-   struct cgroup *old_cgrp, struct task_struct *tsk,
-   bool threadgroup);
+         struct cgroup *old_cgrp, struct task_struct *tsk);
  void (*fork)(struct cgroup_subsys *ss, struct task_struct *task);
  void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
  int (*populate)(struct cgroup_subsys *ss,
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 66a416b..616f27a 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -1750,7 +1750,7 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)

  for_each_subsys(root, ss) {
   if (ss->can_attach) {
-   retval = ss->can_attach(ss, cgrp, tsk, false);
+   retval = ss->can_attach(ss, cgrp, tsk);
    if (retval) {
     /*
      * Remember on which subsystem the can_attach()
@@ -1762,6 +1762,13 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
     goto out;
    }
   }
+  if (ss->can_attach_task) {
+   retval = ss->can_attach_task(cgrp, tsk);
+   if (retval) {
+    failed_ss = ss;
+    goto out;
+   }
+  }
```

```
  }

  task_lock(tsk);
@@ -1798,8 +1805,12 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
  write_unlock(&css_set_lock);

  for_each_subsys(root, ss) {
+  if (ss->pre_attach)
+   ss->pre_attach(cgrp);
+  if (ss->attach_task)
+   ss->attach_task(cgrp, tsk);
   if (ss->attach)
-   ss->attach(ss, cgrp, oldcgrp, tsk, false);
+   ss->attach(ss, cgrp, oldcgrp, tsk);
  }
  set_bit(CGRP_RELEASABLE, &oldcgrp->flags);
  synchronize_rcu();
@@ -1822,7 +1833,7 @@ out:
      */
     break;
    if (ss->cancel_attach)
-    ss->cancel_attach(ss, cgrp, tsk, false);
+    ss->cancel_attach(ss, cgrp, tsk);
   }
  }
  return retval;
diff --git a/kernel/cgroup_freezer.c b/kernel/cgroup_freezer.c
index e7bebb7..e691818 100644
--- a/kernel/cgroup_freezer.c
+++ b/kernel/cgroup_freezer.c
@@ -160,7 +160,7 @@ static void freezer_destroy(struct cgroup_subsys *ss,
  */
 static int freezer_can_attach(struct cgroup_subsys *ss,
          struct cgroup *new_cgroup,
-         struct task_struct *task, bool threadgroup)
+         struct task_struct *task)
 {
  struct freezer *freezer;

@@ -172,26 +172,17 @@ static int freezer_can_attach(struct cgroup_subsys *ss,
  if (freezer->state != CGROUP_THAWED)
   return -EBUSY;

+ return 0;
+}
+
+static int freezer_can_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
+{
```

```
  rcu_read_lock();
- if (__cgroup_freezing_or_frozen(task)) {
+ if (__cgroup_freezing_or_frozen(tsk)) {
   rcu_read_unlock();
   return -EBUSY;
  }
  rcu_read_unlock();
-
- if (threadgroup) {
-  struct task_struct *c;
-
-  rcu_read_lock();
-  list_for_each_entry_rcu(c, &task->thread_group, thread_group) {
-   if (__cgroup_freezing_or_frozen(c)) {
-    rcu_read_unlock();
-    return -EBUSY;
-   }
-  }
-  rcu_read_unlock();
- }
-
  return 0;
 }

@@ -390,6 +381,9 @@ struct cgroup_subsys freezer_subsys = {
  .populate = freezer_populate,
  .subsys_id = freezer_subsys_id,
  .can_attach = freezer_can_attach,
+ .can_attach_task = freezer_can_attach_task,
+ .pre_attach = NULL,
+ .attach_task = NULL,
  .attach  = NULL,
  .fork  = freezer_fork,
  .exit  = NULL,
diff --git a/kernel/cpuset.c b/kernel/cpuset.c
index 4349935..5f71ca2 100644
--- a/kernel/cpuset.c
+++ b/kernel/cpuset.c
@@ -1372,14 +1372,10 @@ static int fmeter_getrate(struct fmeter *fmp)
  return val;
 }

-/* Protected by cgroup_lock */
-static cpumask_var_t cpus_attach;
-
 /* Called by cgroups to determine if a cpuset is usable; cgroup_mutex held */
 static int cpuset_can_attach(struct cgroup_subsys *ss, struct cgroup *cont,
-        struct task_struct *tsk, bool threadgroup)
```

```
+        struct task_struct *tsk)
 {
- int ret;
  struct cpuset *cs = cgroup_cs(cont);

  if (cpumask_empty(cs->cpus_allowed) || nodes_empty(cs->mems_allowed))
@@ -1396,29 +1392,42 @@ static int cpuset_can_attach(struct cgroup_subsys *ss, struct
cgroup *cont,
  if (tsk->flags & PF_THREAD_BOUND)
   return -EINVAL;

- ret = security_task_setscheduler(tsk);
- if (ret)
-  return ret;
- if (threadgroup) {
-  struct task_struct *c;
-
-  rcu_read_lock();
-  list_for_each_entry_rcu(c, &tsk->thread_group, thread_group) {
-   ret = security_task_setscheduler(c);
-   if (ret) {
-    rcu_read_unlock();
-    return ret;
-   }
-  }
-  rcu_read_unlock();
- }
  return 0;
 }

-static void cpuset_attach_task(struct task_struct *tsk, nodemask_t *to,
-         struct cpuset *cs)
+static int cpuset_can_attach_task(struct cgroup *cgrp, struct task_struct *task)
+{
+ return security_task_setscheduler(task);
+}
+
+/*
+ * Protected by cgroup_lock. The nodemasks must be stored globally because
+ * dynamically allocating them is not allowed in pre_attach, and they must
+ * persist among pre_attach, attach_task, and attach.
+ */
+static cpumask_var_t cpus_attach;
+static nodemask_t cpuset_attach_nodemask_from;
+static nodemask_t cpuset_attach_nodemask_to;
+
+/* Set-up work for before attaching each task. */
+static void cpuset_pre_attach(struct cgroup *cont)
```

```
+{
+ struct cpuset *cs = cgroup_cs(cont);
+
+ if (cs == &top_cpuset)
+  cpumask_copy(cpus_attach, cpu_possible_mask);
+ else
+  guarantee_online_cpus(cs, cpus_attach);
+
+ guarantee_online_mems(cs, &cpuset_attach_nodemask_to);
+}
+
+/* Per-thread attachment work. */
+static void cpuset_attach_task(struct cgroup *cont, struct task_struct *tsk)
 {
  int err;
+ struct cpuset *cs = cgroup_cs(cont);
+
  /*
   * can_attach beforehand should guarantee that this doesn't fail.
   * TODO: have a better way to handle failure here
@@ -1426,56 +1435,31 @@ static void cpuset_attach_task(struct task_struct *tsk, nodemask_t
*to,
  err = set_cpus_allowed_ptr(tsk, cpus_attach);
  WARN_ON_ONCE(err);

- cpuset_change_task_nodemask(tsk, to);
+ cpuset_change_task_nodemask(tsk, &cpuset_attach_nodemask_to);
  cpuset_update_task_spread_flag(cs, tsk);
-
 }

 static void cpuset_attach(struct cgroup_subsys *ss, struct cgroup *cont,
-      struct cgroup *oldcont, struct task_struct *tsk,
-      bool threadgroup)
+      struct cgroup *oldcont, struct task_struct *tsk)
 {
  struct mm_struct *mm;
  struct cpuset *cs = cgroup_cs(cont);
  struct cpuset *oldcs = cgroup_cs(oldcont);
- NODEMASK_ALLOC(nodemask_t, from, GFP_KERNEL);
- NODEMASK_ALLOC(nodemask_t, to, GFP_KERNEL);
-
- if (from == NULL || to == NULL)
-  goto alloc_fail;

- if (cs == &top_cpuset) {
-  cpumask_copy(cpus_attach, cpu_possible_mask);
- } else {
```

```
-  guarantee_online_cpus(cs, cpus_attach);
- }
- guarantee_online_mems(cs, to);
-
- /* do per-task migration stuff possibly for each in the threadgroup */
- cpuset_attach_task(tsk, to, cs);
- if (threadgroup) {
-  struct task_struct *c;
-  rcu_read_lock();
-  list_for_each_entry_rcu(c, &tsk->thread_group, thread_group) {
-   cpuset_attach_task(c, to, cs);
-  }
-  rcu_read_unlock();
- }
-
- /* change mm; only needs to be done once even if threadgroup */
- *from = oldcs->mems_allowed;
- *to = cs->mems_allowed;
+ /*
+  * Change mm, possibly for multiple threads in a threadgroup. This is
+  * expensive and may sleep.
+  */
+ cpuset_attach_nodemask_from = oldcs->mems_allowed;
+ cpuset_attach_nodemask_to = cs->mems_allowed;
  mm = get_task_mm(tsk);
  if (mm) {
-  mpol_rebind_mm(mm, to);
+  mpol_rebind_mm(mm, &cpuset_attach_nodemask_to);
   if (is_memory_migrate(cs))
-   cpuset_migrate_mm(mm, from, to);
+   cpuset_migrate_mm(mm, &cpuset_attach_nodemask_from,
+       &cpuset_attach_nodemask_to);
   mmput(mm);
  }
-
-alloc_fail:
- NODEMASK_FREE(from);
- NODEMASK_FREE(to);
 }

 /* The various types of files and directories in a cpuset file system */
@@ -1928,6 +1912,9 @@ struct cgroup_subsys cpuset_subsys = {
  .create = cpuset_create,
  .destroy = cpuset_destroy,
  .can_attach = cpuset_can_attach,
+ .can_attach_task = cpuset_can_attach_task,
+ .pre_attach = cpuset_pre_attach,
+ .attach_task = cpuset_attach_task,
```

```
  .attach = cpuset_attach,
  .populate = cpuset_populate,
  .post_clone = cpuset_post_clone,
diff --git a/kernel/ns_cgroup.c b/kernel/ns_cgroup.c
index 2c98ad9..1fc2b1b 100644
--- a/kernel/ns_cgroup.c
+++ b/kernel/ns_cgroup.c
@@ -43,7 +43,7 @@ int ns_cgroup_clone(struct task_struct *task, struct pid *pid)
  *        ancestor cgroup thereof)
  */
 static int ns_can_attach(struct cgroup_subsys *ss, struct cgroup *new_cgroup,
-    struct task_struct *task, bool threadgroup)
+    struct task_struct *task)
 {
  if (current != task) {
   if (!capable(CAP_SYS_ADMIN))
@@ -53,21 +53,13 @@ static int ns_can_attach(struct cgroup_subsys *ss, struct cgroup
*new_cgroup,
    return -EPERM;
  }

- if (!cgroup_is_descendant(new_cgroup, task))
-  return -EPERM;
-
- if (threadgroup) {
-  struct task_struct *c;
-  rcu_read_lock();
-  list_for_each_entry_rcu(c, &task->thread_group, thread_group) {
-   if (!cgroup_is_descendant(new_cgroup, c)) {
-    rcu_read_unlock();
-    return -EPERM;
-   }
-  }
-  rcu_read_unlock();
- }
+ return 0;
+}

+static int ns_can_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
+{
+ if (!cgroup_is_descendant(cgrp, tsk))
+  return -EPERM;
  return 0;
 }

@@ -112,6 +104,7 @@ static void ns_destroy(struct cgroup_subsys *ss,
 struct cgroup_subsys ns_subsys = {
  .name = "ns",
```

```
  .can_attach = ns_can_attach,
+ .can_attach_task = ns_can_attach_task,
  .create = ns_create,
  .destroy  = ns_destroy,
  .subsys_id = ns_subsys_id,
diff --git a/kernel/sched.c b/kernel/sched.c
index 218ef20..d619f1d 100644
--- a/kernel/sched.c
+++ b/kernel/sched.c
@@ -8655,42 +8655,10 @@ cpu_cgroup_can_attach_task(struct cgroup *cgrp, struct task_struct
*tsk)
  return 0;
 }

-static int
-cpu_cgroup_can_attach(struct cgroup_subsys *ss, struct cgroup *cgrp,
-       struct task_struct *tsk, bool threadgroup)
-{
- int retval = cpu_cgroup_can_attach_task(cgrp, tsk);
- if (retval)
-  return retval;
- if (threadgroup) {
-  struct task_struct *c;
-  rcu_read_lock();
-  list_for_each_entry_rcu(c, &tsk->thread_group, thread_group) {
-   retval = cpu_cgroup_can_attach_task(cgrp, c);
-   if (retval) {
-    rcu_read_unlock();
-    return retval;
-   }
-  }
-  rcu_read_unlock();
- }
- return 0;
-}
-
 static void
-cpu_cgroup_attach(struct cgroup_subsys *ss, struct cgroup *cgrp,
-    struct cgroup *old_cont, struct task_struct *tsk,
-    bool threadgroup)
+cpu_cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
 {
  sched_move_task(tsk);
- if (threadgroup) {
-  struct task_struct *c;
-  rcu_read_lock();
-  list_for_each_entry_rcu(c, &tsk->thread_group, thread_group) {
-   sched_move_task(c);
```

```
- }
-  rcu_read_unlock();
- }
 }

 #ifdef CONFIG_FAIR_GROUP_SCHED
@@ -8763,8 +8731,8 @@ struct cgroup_subsys cpu_cgroup_subsys = {
  .name  = "cpu",
  .create  = cpu_cgroup_create,
  .destroy = cpu_cgroup_destroy,
- .can_attach = cpu_cgroup_can_attach,
- .attach  = cpu_cgroup_attach,
+ .can_attach_task = cpu_cgroup_can_attach_task,
+ .attach_task = cpu_cgroup_attach_task,
  .populate = cpu_cgroup_populate,
  .subsys_id = cpu_cgroup_subsys_id,
  .early_init = 1,
diff --git a/mm/memcontrol.c b/mm/memcontrol.c
index 729beb7..995f0b9 100644
--- a/mm/memcontrol.c
+++ b/mm/memcontrol.c
@@ -4720,8 +4720,7 @@ static void mem_cgroup_clear_mc(void)

 static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
     struct cgroup *cgroup,
-    struct task_struct *p,
-    bool threadgroup)
+    struct task_struct *p)
 {
  int ret = 0;
  struct mem_cgroup *mem = mem_cgroup_from_cont(cgroup);
@@ -4775,8 +4774,7 @@ static int mem_cgroup_can_attach(struct cgroup_subsys *ss,

 static void mem_cgroup_cancel_attach(struct cgroup_subsys *ss,
     struct cgroup *cgroup,
-    struct task_struct *p,
-    bool threadgroup)
+    struct task_struct *p)
 {
  mem_cgroup_clear_mc();
 }
@@ -4880,8 +4878,7 @@ static void mem_cgroup_move_charge(struct mm_struct *mm)
 static void mem_cgroup_move_task(struct cgroup_subsys *ss,
     struct cgroup *cont,
     struct cgroup *old_cont,
-    struct task_struct *p,
-    bool threadgroup)
+    struct task_struct *p)
```

```
 {
  if (!mc.mm)
   /* no need to move charge */
@@ -4893,22 +4890,19 @@ static void mem_cgroup_move_task(struct cgroup_subsys *ss,
 #else /* !CONFIG_MMU */
 static int mem_cgroup_can_attach(struct cgroup_subsys *ss,
     struct cgroup *cgroup,
-    struct task_struct *p,
-    bool threadgroup)
+    struct task_struct *p)
 {
  return 0;
 }
 static void mem_cgroup_cancel_attach(struct cgroup_subsys *ss,
     struct cgroup *cgroup,
-    struct task_struct *p,
-    bool threadgroup)
+    struct task_struct *p)
 {
 }
 static void mem_cgroup_move_task(struct cgroup_subsys *ss,
     struct cgroup *cont,
     struct cgroup *old_cont,
-    struct task_struct *p,
-    bool threadgroup)
+    struct task_struct *p)
 {
 }
 #endif
diff --git a/security/device_cgroup.c b/security/device_cgroup.c
index 8d9c48f..cd1f779 100644
--- a/security/device_cgroup.c
+++ b/security/device_cgroup.c
@@ -62,8 +62,7 @@ static inline struct dev_cgroup *task_devcgroup(struct task_struct *task)
 struct cgroup_subsys devices_subsys;

 static int devcgroup_can_attach(struct cgroup_subsys *ss,
-  struct cgroup *new_cgroup, struct task_struct *task,
-  bool threadgroup)
+  struct cgroup *new_cgroup, struct task_struct *task)
 {
  if (current != task && !capable(CAP_SYS_ADMIN))
   return -EPERM;
```

_____

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