Subject: [RFC] [PATCH] Re: Prefixing cgroup generic control filenames with "cgroup."
Posted by Paul Menage on Fri, 29 Feb 2008 05:59:21 GMT
View Forum Message <> Reply to Message

>> to something like:
>> >
>> > /mnt/cgroup/
>> > tasks
>> > cpu.shares
>> > memory.limit_in_bytes
>> > memory.usage_in_bytes
>> > groups/
>> > user_created_groupname1/
>> > tasks
>> > cpu.shares
>> > memory.limit_in_bytes
>> > memory.usage_in_bytes
>> > groups/
>> > user_created_groupname2/
>> > tasks
>> > cpu.shares
>> > memory.limit_in_bytes
>> > memory.usage_in_bytes
>> > groups/
>
> That looks nice.
>

OK, here's a patch that does that. It's not quite right yet, as it crashes on unmount, but the basic idea is there. It adds the additional "groups" directory by default, but uses the previous behaviour if the nogroupdir option is passed (done by default for cpusets).

Thoughts? Is this a direction we want to go in? As an option, or by default?

Paul

```
 include/linux/cgroup.h |    2
 kernel/cgroup.c        |  164 ++++++++++++++++++++++++++++++++++++++----------
 kernel/cpuset.c        |    2
 3 files changed, 122 insertions(+), 46 deletions(-)

Index: subdir-2.6.25-rc3/include/linux/cgroup.h
===================================================================
--- subdir-2.6.25-rc3.orig/include/linux/cgroup.h
+++ subdir-2.6.25-rc3/include/linux/cgroup.h
@@ -105,7 +105,7 @@ struct cgroup {
```

```
  struct cgroup *parent; /* my parent */
  struct dentry *dentry;    /* cgroup fs entry */
-
+ struct dentry *group_dentry;
  /* Private pointers for each registered subsystem */
  struct cgroup_subsys_state *subsys[CGROUP_SUBSYS_COUNT];

Index: subdir-2.6.25-rc3/kernel/cgroup.c
===================================================================
--- subdir-2.6.25-rc3.orig/kernel/cgroup.c
+++ subdir-2.6.25-rc3/kernel/cgroup.c
@@ -139,6 +139,7 @@ inline int cgroup_is_removed(const struc
 /* bits in struct cgroupfs_root flags field */
 enum {
  ROOT_NOPREFIX, /* mounted subsystems have no named prefix */
+ ROOT_NOGROUPDIR, /* user-created sub-groups go in the main directory */
 };

 static int cgroup_is_releasable(const struct cgroup *cgrp)
@@ -475,6 +476,16 @@ static struct css_set *find_css_set(
  return res;
 }

+static inline struct cgroup *__d_cgrp(struct dentry *dentry)
+{
+ return dentry->d_fsdata;
+}
+
+static inline struct cftype *__d_cft(struct dentry *dentry)
+{
+ return dentry->d_fsdata;
+}
+
 /*
  * There is one global cgroup mutex. We also require taking
  * task_lock() when dereferencing a task's cgroup subsys pointers.
@@ -598,33 +609,41 @@ static void cgroup_diput(struct dentry *
  /* is dentry a directory ? if so, kfree() associated cgroup */
  if (S_ISDIR(inode->i_mode)) {
   struct cgroup *cgrp = dentry->d_fsdata;
-  struct cgroup_subsys *ss;
-  BUG_ON(!(cgroup_is_removed(cgrp)));
-  /* It's possible for external users to be holding css
-   * reference counts on a cgroup; css_put() needs to
-   * be able to access the cgroup after decrementing
-   * the reference count in order to know if it needs to
-   * queue the cgroup to be handled by the release
```

```
-   * agent */
-  synchronize_rcu();
+  if (dentry != cgrp->group_dentry) {
+   struct cgroup_subsys *ss;
+   BUG_ON(!(cgroup_is_removed(cgrp)));
+   /*
+    * It's possible for external users to be
+    * holding css reference counts on a cgroup;
+    * css_put() needs to be able to access the
+    * cgroup after decrementing the reference
+    * count in order to know if it needs to queue
+    * the cgroup to be handled by the release
+    * agent
+    */
+   synchronize_rcu();

-  mutex_lock(&cgroup_mutex);
-  /*
-   * Release the subsystem state objects.
-   */
-  for_each_subsys(cgrp->root, ss) {
-   if (cgrp->subsys[ss->subsys_id])
-    ss->destroy(ss, cgrp);
-  }
+   mutex_lock(&cgroup_mutex);
+   /*
+    * Release the subsystem state objects.
+    */
+   for_each_subsys(cgrp->root, ss) {
+    if (cgrp->subsys[ss->subsys_id])
+     ss->destroy(ss, cgrp);
+   }

-  cgrp->root->number_of_cgroups--;
-  mutex_unlock(&cgroup_mutex);
+   cgrp->root->number_of_cgroups--;
+   mutex_unlock(&cgroup_mutex);

-  /* Drop the active superblock reference that we took when we
-   * created the cgroup */
-  deactivate_super(cgrp->root->sb);
+   /*
+    * Drop the active superblock reference that
+    * we took when we created the cgroup
+    */
+   deactivate_super(cgrp->root->sb);

-  kfree(cgrp);
```

```
+   kfree(cgrp);
+  } else
+   cgrp->group_dentry = NULL;
  }
  iput(inode);
 }
@@ -638,24 +657,40 @@ static void remove_dir(struct dentry *d)
  dput(parent);
 }

-static void cgroup_clear_directory(struct dentry *dentry)
+static void cgroup_d_remove_dir(struct dentry *dentry);
+
+static void cgroup_clear_directory(struct dentry *dentry, int zap_groupdir)
 {
  struct list_head *node;
-
+ struct cgroup *cgrp;
  BUG_ON(!mutex_is_locked(&dentry->d_inode->i_mutex));
  spin_lock(&dcache_lock);
+ cgrp = __d_cgrp(dentry);
  node = dentry->d_subdirs.next;
  while (node != &dentry->d_subdirs) {
   struct dentry *d = list_entry(node, struct dentry, d_u.d_child);
+  if ((d == cgrp->group_dentry) && !zap_groupdir) {
+   node = node->next;
+   continue;
+  }
+
   list_del_init(node);
   if (d->d_inode) {
-   /* This should never be called on a cgroup
-    * directory with child cgroups */
-   BUG_ON(d->d_inode->i_mode & S_IFDIR);
    d = dget_locked(d);
    spin_unlock(&dcache_lock);
    d_delete(d);
-   simple_unlink(dentry->d_inode, d);
+   if (d == cgrp->group_dentry) {
+    mutex_lock(&d->d_inode->i_mutex);
+    cgroup_d_remove_dir(d);
+    mutex_unlock(&d->d_inode->i_mutex);
+    dput(dentry);
+    cgrp->group_dentry = NULL;
+   } else {
+    /* This should never be called on a cgroup
+     * directory with child cgroups */
+    BUG_ON(d->d_inode->i_mode & S_IFDIR);
```

```
+    simple_unlink(dentry->d_inode, d);
+   }
    dput(d);
    spin_lock(&dcache_lock);
   }
@@ -669,12 +704,13 @@ static void cgroup_clear_directory(struc
  */
 static void cgroup_d_remove_dir(struct dentry *dentry)
 {
- cgroup_clear_directory(dentry);
+ cgroup_clear_directory(dentry, 1);

  spin_lock(&dcache_lock);
  list_del_init(&dentry->d_u.d_child);
  spin_unlock(&dcache_lock);
  remove_dir(dentry);
+ dput(dentry);
 }

 static int rebind_subsystems(struct cgroupfs_root *root,
@@ -755,6 +791,8 @@ static int cgroup_show_options(struct se
   seq_printf(seq, ",%s", ss->name);
  if (test_bit(ROOT_NOPREFIX, &root->flags))
   seq_puts(seq, ",noprefix");
+ if (test_bit(ROOT_NOGROUPDIR, &root->flags))
+  seq_puts(seq, ",nogroupdir");
  if (strlen(root->release_agent_path))
   seq_printf(seq, ",release_agent=%s", root->release_agent_path);
  mutex_unlock(&cgroup_mutex);
@@ -785,6 +823,8 @@ static int parse_cgroupfs_options(char *
    opts->subsys_bits = (1 << CGROUP_SUBSYS_COUNT) - 1;
   } else if (!strcmp(token, "noprefix")) {
    set_bit(ROOT_NOPREFIX, &opts->flags);
+  } else if (!strcmp(token, "nogroupdir")) {
+   set_bit(ROOT_NOGROUPDIR, &opts->flags);
   } else if (!strncmp(token, "release_agent=", 14)) {
    /* Specifying two release agents is forbidden */
    if (opts->release_agent)
@@ -932,6 +972,8 @@ static int cgroup_get_rootdir(struct sup
  return 0;
 }

+static int cgroup_create_groupdir(struct cgroup *cgrp, int mode);
+
 static int cgroup_get_sb(struct file_system_type *fs_type,
     int flags, const char *unused_dev_name,
     void *data, struct vfsmount *mnt)
@@ -1050,6 +1092,8 @@ static int cgroup_get_sb(struct file_sys
```

```
   BUG_ON(!list_empty(&cgrp->children));
   BUG_ON(root->number_of_cgroups != 1);

+  cgroup_create_groupdir(cgrp, 0755);
+
   cgroup_populate_dir(cgrp);
   mutex_unlock(&inode->i_mutex);
   mutex_unlock(&cgroup_mutex);
@@ -1103,6 +1147,10 @@ static void cgroup_kill_sb(struct super_
  }
  mutex_unlock(&cgroup_mutex);

+ if (cgrp->group_dentry) {
+  dput(cgrp->group_dentry);
+  cgrp->group_dentry = NULL;
+ }
  kfree(root);
  kill_litter_super(sb);
 }
@@ -1113,16 +1161,6 @@ static struct file_system_type cgroup_fs
  .kill_sb = cgroup_kill_sb,
 };

-static inline struct cgroup *__d_cgrp(struct dentry *dentry)
-{
- return dentry->d_fsdata;
-}
-
-static inline struct cftype *__d_cft(struct dentry *dentry)
-{
- return dentry->d_fsdata;
-}
-
 /**
  * cgroup_path - generate the path of a cgroup
  * @cgrp: the cgroup in question
@@ -1538,13 +1576,17 @@ static struct file_operations cgroup_fil
  .release = cgroup_file_release,
 };

-static struct inode_operations cgroup_dir_inode_operations = {
+static struct inode_operations cgroup_groupdir_inode_operations = {
  .lookup = simple_lookup,
  .mkdir = cgroup_mkdir,
  .rmdir = cgroup_rmdir,
  .rename = cgroup_rename,
 };
```

```
+static struct inode_operations cgroup_dir_inode_operations = {
+ .lookup = simple_lookup,
+};
+
 static int cgroup_create_file(struct dentry *dentry, int mode,
     struct super_block *sb)
 {
@@ -1609,6 +1651,39 @@ static int cgroup_create_dir(struct cgro
  return error;
 }

+static int cgroup_create_groupdir(struct cgroup *cgrp, int mode)
+{
+ struct dentry *parent, *dentry;
+ int error = 0;
+
+ parent = cgrp->dentry;
+
+ if (test_bit(ROOT_NOGROUPDIR, &cgrp->root->flags)) {
+  parent->d_inode->i_op = &cgroup_groupdir_inode_operations;
+  return 0;
+ }
+
+ dentry = lookup_one_len("groups", parent, strlen("groups"));
+ if (!IS_ERR(dentry)) {
+  error = cgroup_create_file(dentry,
+        S_IFDIR | mode, cgrp->root->sb);
+  if (!error) {
+   dentry->d_fsdata = cgrp;
+   dentry->d_inode->i_op =
+    &cgroup_groupdir_inode_operations;
+   inc_nlink(parent->d_inode);
+   cgrp->group_dentry = dentry;
+   dget(dentry);
+   mutex_unlock(&dentry->d_inode->i_mutex);
+  }
+  dput(dentry);
+ } else {
+  error = PTR_ERR(dentry);
+ }
+
+ return error;
+}
+
 int cgroup_add_file(struct cgroup *cgrp,
         struct cgroup_subsys *subsys,
         const struct cftype *cft)
@@ -2169,8 +2244,8 @@ static int cgroup_populate_dir(struct cg
```

```
  int err;
  struct cgroup_subsys *ss;

- /* First clear out any existing files */
- cgroup_clear_directory(cgrp->dentry);
+ /* First clear out any existing control files */
+ cgroup_clear_directory(cgrp->dentry, 0);

  err = cgroup_add_files(cgrp, NULL, files, ARRAY_SIZE(files));
  if (err < 0)
@@ -2258,9 +2333,11 @@ static long cgroup_create(struct cgroup
  if (err < 0)
   goto err_remove;

+
  /* The cgroup directory was pre-locked for us */
  BUG_ON(!mutex_is_locked(&cgrp->dentry->d_inode->i_mutex));

+ cgroup_create_groupdir(cgrp, mode);
  err = cgroup_populate_dir(cgrp);
  /* If err < 0, we have a half-filled directory - oh well ;) */

@@ -2377,7 +2454,6 @@ static int cgroup_rmdir(struct inode *un
  spin_unlock(&d->d_lock);

  cgroup_d_remove_dir(d);
- dput(d);

  set_bit(CGRP_RELEASABLE, &parent->flags);
  check_for_release(parent);
Index: subdir-2.6.25-rc3/kernel/cpuset.c
===================================================================
--- subdir-2.6.25-rc3.orig/kernel/cpuset.c
+++ subdir-2.6.25-rc3/kernel/cpuset.c
@@ -243,7 +243,7 @@ static int cpuset_get_sb(struct file_sys
  int ret = -ENODEV;
  if (cgroup_fs) {
   char mountopts[] =
-   "cpuset,noprefix,"
+   "cpuset,noprefix,nogroupdir"
    "release_agent=/sbin/cpuset_release_agent";
   ret = cgroup_fs->get_sb(cgroup_fs, flags,
        unused_dev_name, mountopts, mnt);
```