
Subject: [PATCH] Control groups: Replace "cont" with "cgrp" and other misc renaming

Posted by [Paul Menage](#) on Tue, 16 Oct 2007 11:32:59 GMT

[View Forum Message](#) <> [Reply to Message](#)

Replace "cont" with "cgrp" and other misc renaming

This patch finishes some of the names that got missed in the great "task containers" -> "control groups" rename. Primarily it renames the local variable "cont" to "cgrp" in a number of places, and renames the CONT_* enum members to CGRP_*.

This patch is not intended to have any effect on the generated code; the output of "objdump -d kernel/cgroup.o" is unchanged.

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

kernel/cgroup.c | 496 ++++++-----
1 file changed, 248 insertions(+), 248 deletions(-)

Index: container-2.6.23-mm1/kernel/cgroup.c

=====

--- container-2.6.23-mm1.orig/kernel/cgroup.c

+++ container-2.6.23-mm1/kernel/cgroup.c

@@ -123,18 +123,18 @@ static int need_forkexit_callback;

/* bits in struct cgroup flags field */

enum {

/* Control Group is dead */

- CONT_REMOVED,

+ CGRP_REMOVED,

/* Control Group has previously had a child cgroup or a task,

- * but no longer (only if CONT_NOTIFY_ON_RELEASE is set) */

- CONT_RELEASABLE,

+ * but no longer (only if CGRP_NOTIFY_ON_RELEASE is set) */

+ CGRP_RELEASABLE,

/* Control Group requires release notifications to userspace */

- CONT_NOTIFY_ON_RELEASE,

+ CGRP_NOTIFY_ON_RELEASE,

};

/* convenient tests for these bits */

-inline int cgroup_is_removed(const struct cgroup *cont)

+inline int cgroup_is_removed(const struct cgroup *cgrp)

{

- return test_bit(CONT_REMOVED, &cont->flags);

+ return test_bit(CGRP_REMOVED, &cgrp->flags);

}

```

/* bits in struct cgroupfs_root flags field */
@@ -142,17 +142,17 @@ enum {
    ROOT_NOPREFIX, /* mounted subsystems have no named prefix */
};

-inline int cgroup_is_releasable(const struct cgroup *cont)
+inline int cgroup_is_releasable(const struct cgroup *cgrp)
{
    const int bits =
- (1 << CONT_RELEASABLE) |
- (1 << CONT_NOTIFY_ON_RELEASE);
- return (cont->flags & bits) == bits;
+ (1 << CGRP_RELEASABLE) |
+ (1 << CGRP_NOTIFY_ON_RELEASE);
+ return (cgrp->flags & bits) == bits;
}

-inline int notify_on_release(const struct cgroup *cont)
+inline int notify_on_release(const struct cgroup *cgrp)
{
- return test_bit(CONT_NOTIFY_ON_RELEASE, &cont->flags);
+ return test_bit(CGRP_NOTIFY_ON_RELEASE, &cgrp->flags);
}

/*
@@ -172,7 +172,7 @@ static LIST_HEAD(release_list);
static DEFINE_SPINLOCK(release_list_lock);
static void cgroup_release_agent(struct work_struct *work);
static DECLARE_WORK(release_agent_work, cgroup_release_agent);
-static void check_for_release(struct cgroup *cont);
+static void check_for_release(struct cgroup *cgrp);

/* Link structure for associating css_set objects with cgroups */
struct cg_cgroup_link {
@@ -180,7 +180,7 @@ struct cg_cgroup_link {
    * List running through cg_cgroup_links associated with a
    * cgroup, anchored on cgroup->css_sets
    */
- struct list_head cont_link_list;
+ struct list_head cgrp_link_list;
}

/*
    * List running through cg_cgroup_links pointing at a
    * single css_set object, anchored on css_set->cg_links
@@ -238,7 +238,7 @@ static void unlink_css_set(struct css_se
    link = list_entry(cg->cg_links.next,
        struct cg_cgroup_link, cg_link_list);
    list_del(&link->cg_link_list);

```

```

- list_del(&link->cont_link_list);
+ list_del(&link->cgrp_link_list);
  kfree(link);
}
  write_unlock(&css_set_lock);
@@ -253,12 +253,12 @@ static void __release_css_set(struct kre

  rcu_read_lock();
  for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
- struct cgroup *cont = cg->subsys[i]->cgroup;
- if (atomic_dec_and_test(&cont->count) &&
-     notify_on_release(cont)) {
+ struct cgroup *cgrp = cg->subsys[i]->cgroup;
+ if (atomic_dec_and_test(&cgrp->count) &&
+     notify_on_release(cgrp)) {
    if (taskexit)
-   set_bit(CONT_RELEASABLE, &cont->flags);
-   check_for_release(cont);
+   set_bit(CGRP_RELEASABLE, &cgrp->flags);
+   check_for_release(cgrp);
  }
}
  rcu_read_unlock();
@@ -303,7 +303,7 @@ static inline void put_css_set_taskexit(
 * oldcg: the cgroup group that we're using before the cgroup
 * transition
 *
- * cont: the cgroup that we're moving into
+ * cgrp: the cgroup that we're moving into
 *
 * template: location in which to build the desired set of subsystem
 * state objects for the new cgroup group
@@ -311,11 +311,11 @@ static inline void put_css_set_taskexit(

static struct css_set *find_existing_css_set(
  struct css_set *oldcg,
- struct cgroup *cont,
+ struct cgroup *cgrp,
  struct cgroup_subsys_state *template[])
{
  int i;
- struct cgroupfs_root *root = cont->root;
+ struct cgroupfs_root *root = cgrp->root;
  struct list_head *l = &init_css_set.list;

  /* Built the set of subsystem state objects that we want to
@@ -325,7 +325,7 @@ static struct css_set *find_existing_css
  /* Subsystem is in this hierarchy. So we want

```

```

    * the subsystem state from the new
    * cgroup */
- template[i] = cont->subsys[i];
+ template[i] = cgrp->subsys[i];
} else {
    /* Subsystem is not in this hierarchy, so we
    * don't want to change the subsystem state */
@@ -352,7 +352,7 @@ static struct css_set *find_existing_css

/*
 * allocate_cg_links() allocates "count" cg_cgroup_link structures
- * and chains them on tmp through their cont_link_list fields. Returns 0 on
+ * and chains them on tmp through their cgrp_link_list fields. Returns 0 on
 * success or a negative error
 */

@@ -367,13 +367,13 @@ static int allocate_cg_links(int count,
while (!list_empty(tmp)) {
    link = list_entry(tmp->next,
        struct cg_cgroup_link,
-    cont_link_list);
- list_del(&link->cont_link_list);
+    cgrp_link_list);
+ list_del(&link->cgrp_link_list);
    kfree(link);
}
return -ENOMEM;
}
- list_add(&link->cont_link_list, tmp);
+ list_add(&link->cgrp_link_list, tmp);
}
return 0;
}
@@ -384,8 +384,8 @@ static void free_cg_links(struct list_he
struct cg_cgroup_link *link;
link = list_entry(tmp->next,
    struct cg_cgroup_link,
-    cont_link_list);
- list_del(&link->cont_link_list);
+    cgrp_link_list);
+ list_del(&link->cgrp_link_list);
    kfree(link);
}
}
@@ -399,7 +399,7 @@ static void free_cg_links(struct list_he
*/

static struct css_set *find_css_set(

```

```

- struct css_set *oldcg, struct cgroup *cont)
+ struct css_set *oldcg, struct cgroup *cgrp)
{
    struct css_set *res;
    struct cgroup_subsys_state *template[CGROUP_SUBSYS_COUNT];
@@ -411,7 +411,7 @@ static struct css_set *find_css_set(
    /* First see if we already have a cgroup group that matches
     * the desired set */
    write_lock(&css_set_lock);
- res = find_existing_css_set(oldcg, cont, template);
+ res = find_existing_css_set(oldcg, cgrp, template);
    if (res)
        get_css_set(res);
    write_unlock(&css_set_lock);
@@ -440,9 +440,9 @@ static struct css_set *find_css_set(
    write_lock(&css_set_lock);
    /* Add reference counts and links from the new css_set. */
    for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
- struct cgroup *cont = res->subsys[i]->cgroup;
+ struct cgroup *cgrp = res->subsys[i]->cgroup;
        struct cgroup_subsys *ss = subsys[i];
- atomic_inc(&cont->count);
+ atomic_inc(&cgrp->count);
    /*
     * We want to add a link once per cgroup, so we
     * only do it for the first subsystem in each
@@ -452,9 +452,9 @@ static struct css_set *find_css_set(
    BUG_ON(list_empty(&tmp_cg_links));
    link = list_entry(tmp_cg_links.next,
        struct cg_cgroup_link,
-    cont_link_list);
- list_del(&link->cont_link_list);
- list_add(&link->cont_link_list, &cont->css_sets);
+    cgrp_link_list);
+ list_del(&link->cgrp_link_list);
+ list_add(&link->cgrp_link_list, &cgrp->css_sets);
    link->cg = res;
    list_add(&link->cg_link_list, &res->cg_links);
}
@@ -462,9 +462,9 @@ static struct css_set *find_css_set(
    if (list_empty(&rootnode.subsys_list)) {
        link = list_entry(tmp_cg_links.next,
            struct cg_cgroup_link,
-        cont_link_list);
- list_del(&link->cont_link_list);
- list_add(&link->cont_link_list, &dummysys->css_sets);
+        cgrp_link_list);
+ list_del(&link->cgrp_link_list);

```

```

+ list_add(&link->cgrp_link_list, &dummytop->css_sets);
  link->cg = res;
  list_add(&link->cg_link_list, &res->cg_links);
}
@@ -564,7 +564,7 @@ void cgroup_unlock(void)

static int cgroup_mkdir(struct inode *dir, struct dentry *dentry, int mode);
static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry);
-static int cgroup_populate_dir(struct cgroup *cont);
+static int cgroup_populate_dir(struct cgroup *cgrp);
static struct inode_operations cgroup_dir_inode_operations;
static struct file_operations proc_cgroupstats_operations;

@@ -591,8 +591,8 @@ static void cgroup_diput(struct dentry *
{
  /* is dentry a directory ? if so, kfree() associated cgroup */
  if (S_ISDIR(inode->i_mode)) {
-   struct cgroup *cont = dentry->d_fsdata;
-   BUG_ON(!(cgroup_is_removed(cont)));
+   struct cgroup *cgrp = dentry->d_fsdata;
+   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
@@ -600,7 +600,7 @@ static void cgroup_diput(struct dentry *
    * queue the cgroup to be handled by the release
    * agent */
    synchronize_rcu();
-   kfree(cont);
+   kfree(cgrp);
  }
  iput(inode);
}
@@ -657,7 +657,7 @@ static int rebind_subsystems(struct cgro
    unsigned long final_bits)
{
  unsigned long added_bits, removed_bits;
-   struct cgroup *cont = &root->top_cgroup;
+   struct cgroup *cgrp = &root->top_cgroup;
  int i;

  removed_bits = root->actual_subsys_bits & ~final_bits;
@@ -678,7 +678,7 @@ static int rebind_subsystems(struct cgro
    * any child cgroups exist. This is theoretically supportable
    * but involves complex error handling, so it's being left until
    * later */
-   if (!list_empty(&cont->children))
+   if (!list_empty(&cgrp->children))

```

```

return -EBUSY;

/* Process each subsystem */
@@ -687,32 +687,32 @@ static int rebind_subsystems(struct cgr
    unsigned long bit = 1UL << i;
    if (bit & added_bits) {
        /* We're binding this subsystem to this hierarchy */
-       BUG_ON(cont->subsys[i]);
+       BUG_ON(cgrp->subsys[i]);
        BUG_ON(!dummytop->subsys[i]);
        BUG_ON(dummytop->subsys[i]->cgroup != dummytop);
-       cont->subsys[i] = dummytop->subsys[i];
-       cont->subsys[i]->cgroup = cont;
+       cgrp->subsys[i] = dummytop->subsys[i];
+       cgrp->subsys[i]->cgroup = cgrp;
        list_add(&ss->sibling, &root->subsys_list);
        rcu_assign_pointer(ss->root, root);
        if (ss->bind)
-       ss->bind(ss, cont);
+       ss->bind(ss, cgrp);

    } else if (bit & removed_bits) {
        /* We're removing this subsystem */
-       BUG_ON(cont->subsys[i] != dummytop->subsys[i]);
-       BUG_ON(cont->subsys[i]->cgroup != cont);
+       BUG_ON(cgrp->subsys[i] != dummytop->subsys[i]);
+       BUG_ON(cgrp->subsys[i]->cgroup != cgrp);
        if (ss->bind)
            ss->bind(ss, dummytop);
        dummytop->subsys[i]->cgroup = dummytop;
-       cont->subsys[i] = NULL;
+       cgrp->subsys[i] = NULL;
        rcu_assign_pointer(subsys[i]->root, &rootnode);
        list_del(&ss->sibling);
    } else if (bit & final_bits) {
        /* Subsystem state should already exist */
-       BUG_ON(!cont->subsys[i]);
+       BUG_ON(!cgrp->subsys[i]);
    } else {
        /* Subsystem state shouldn't exist */
-       BUG_ON(cont->subsys[i]);
+       BUG_ON(cgrp->subsys[i]);
    }
}
root->subsys_bits = root->actual_subsys_bits = final_bits;
@@ -796,10 +796,10 @@ static int cgroup_remount(struct super_b
{
    int ret = 0;

```

```

    struct cgroupfs_root *root = sb->s_fs_info;
- struct cgroup *cont = &root->top_cgroup;
+ struct cgroup *cgrp = &root->top_cgroup;
    struct cgroup_sb_opts opts;

- mutex_lock(&cont->dentry->d_inode->i_mutex);
+ mutex_lock(&cgrp->dentry->d_inode->i_mutex);
    mutex_lock(&cgroup_mutex);

    /* See what subsystems are wanted */
@@ -817,7 +817,7 @@ static int cgroup_remount(struct super_b

    /* (re)populate subsystem files */
    if (!ret)
- cgroup_populate_dir(cont);
+ cgroup_populate_dir(cgrp);

    if (opts.release_agent)
        strcpy(root->release_agent_path, opts.release_agent);
@@ -825,7 +825,7 @@ static int cgroup_remount(struct super_b
    if (opts.release_agent)
        kfree(opts.release_agent);
    mutex_unlock(&cgroup_mutex);
- mutex_unlock(&cont->dentry->d_inode->i_mutex);
+ mutex_unlock(&cgrp->dentry->d_inode->i_mutex);
    return ret;
}

@@ -838,16 +838,16 @@ static struct super_operations cgroup_op

static void init_cgroup_root(struct cgroupfs_root *root)
{
- struct cgroup *cont = &root->top_cgroup;
+ struct cgroup *cgrp = &root->top_cgroup;
    INIT_LIST_HEAD(&root->subsys_list);
    INIT_LIST_HEAD(&root->root_list);
    root->number_of_cgroups = 1;
- cont->root = root;
- cont->top_cgroup = cont;
- INIT_LIST_HEAD(&cont->sibling);
- INIT_LIST_HEAD(&cont->children);
- INIT_LIST_HEAD(&cont->css_sets);
- INIT_LIST_HEAD(&cont->release_list);
+ cgrp->root = root;
+ cgrp->top_cgroup = cgrp;
+ INIT_LIST_HEAD(&cgrp->sibling);
+ INIT_LIST_HEAD(&cgrp->children);
+ INIT_LIST_HEAD(&cgrp->css_sets);

```



```

+ INIT_LIST_HEAD(&cgrp->release_list);
}

static int cgroup_test_super(struct super_block *sb, void *data)
@@ -954,7 +954,7 @@ static int cgroup_get_sb(struct file_sys
    root = NULL;
} else {
    /* New superblock */
- struct cgroup *cont = &root->top_cgroup;
+ struct cgroup *cgrp = &root->top_cgroup;
    struct inode *inode;

    BUG_ON(sb->s_root != NULL);
@@ -1008,10 +1008,10 @@ static int cgroup_get_sb(struct file_sys
    BUG_ON(list_empty(&tmp_cg_links));
    link = list_entry(tmp_cg_links.next,
        struct cg_cgroup_link,
-    cont_link_list);
- list_del(&link->cont_link_list);
+    cgrp_link_list);
+ list_del(&link->cgrp_link_list);
    link->cg = cg;
- list_add(&link->cont_link_list,
+ list_add(&link->cgrp_link_list,
        &root->top_cgroup.css_sets);
    list_add(&link->cg_link_list, &cg->cg_links);
    l = l->next;
@@ -1020,11 +1020,11 @@ static int cgroup_get_sb(struct file_sys

    free_cg_links(&tmp_cg_links);

- BUG_ON(!list_empty(&cont->sibling));
- BUG_ON(!list_empty(&cont->children));
+ BUG_ON(!list_empty(&cgrp->sibling));
+ BUG_ON(!list_empty(&cgrp->children));
    BUG_ON(root->number_of_cgroups != 1);

- cgroup_populate_dir(cont);
+ cgroup_populate_dir(cgrp);
    mutex_unlock(&inode->i_mutex);
    mutex_unlock(&cgroup_mutex);
}
@@ -1040,14 +1040,14 @@ static int cgroup_get_sb(struct file_sys

static void cgroup_kill_sb(struct super_block *sb) {
    struct cgroupfs_root *root = sb->s_fs_info;
- struct cgroup *cont = &root->top_cgroup;
+ struct cgroup *cgrp = &root->top_cgroup;

```

```

int ret;

BUG_ON(!root);

BUG_ON(root->number_of_cgroups != 1);
- BUG_ON(!list_empty(&cont->children));
- BUG_ON(!list_empty(&cont->sibling));
+ BUG_ON(!list_empty(&cgrp->children));
+ BUG_ON(!list_empty(&cgrp->sibling));

mutex_lock(&cgroup_mutex);

@@ -1061,12 +1061,12 @@ static void cgroup_kill_sb(struct super_
* root cgroup
*/
write_lock(&css_set_lock);
- while (!list_empty(&cont->css_sets)) {
+ while (!list_empty(&cgrp->css_sets)) {
    struct cg_cgroup_link *link;
-   link = list_entry(cont->css_sets.next,
-       struct cg_cgroup_link, cont_link_list);
+   link = list_entry(cgrp->css_sets.next,
+       struct cg_cgroup_link, cgrp_link_list);
    list_del(&link->cg_link_list);
-   list_del(&link->cont_link_list);
+   list_del(&link->cgrp_link_list);
    kfree(link);
}
write_unlock(&css_set_lock);
@@ -1087,7 +1087,7 @@ static struct file_system_type cgroup_fs
.kill_sb = cgroup_kill_sb,
};

-static inline struct cgroup *__d_cont(struct dentry *dentry)
+static inline struct cgroup *__d_cgrp(struct dentry *dentry)
{
    return dentry->d_fsdata;
}
@@ -1101,11 +1101,11 @@ static inline struct cftype *__d_cft(str
* Called with cgroup_mutex held. Writes path of cgroup into buf.
* Returns 0 on success, -errno on error.
*/
-int cgroup_path(const struct cgroup *cont, char *buf, int buflen)
+int cgroup_path(const struct cgroup *cgrp, char *buf, int buflen)
{
    char *start;

- if (cont == dummytop) {

```

```

+ if (cgrp == dummytop) {
/*
 * Inactive subsystems have no dentry for their root
 * cgroup
@@ -1118,14 +1118,14 @@ int cgroup_path(const struct cgroup *con

    *--start = '\0';
    for (;;) {
- int len = cont->dentry->d_name.len;
+ int len = cgrp->dentry->d_name.len;
        if ((start -= len) < buf)
            return -ENAMETOOLONG;
- memcpy(start, cont->dentry->d_name.name, len);
- cont = cont->parent;
- if (!cont)
+ memcpy(start, cgrp->dentry->d_name.name, len);
+ cgrp = cgrp->parent;
+ if (!cgrp)
            break;
- if (!cont->parent)
+ if (!cgrp->parent)
            continue;
        if (--start < buf)
            return -ENAMETOOLONG;
@@ -1140,16 +1140,16 @@ int cgroup_path(const struct cgroup *con
    * its subsystem id.
    */

-static void get_first_subsys(const struct cgroup *cont,
+static void get_first_subsys(const struct cgroup *cgrp,
    struct cgroup_subsys_state **css, int *subsys_id)
{
- const struct cgroupfs_root *root = cont->root;
+ const struct cgroupfs_root *root = cgrp->root;
    const struct cgroup_subsys *test_ss;
    BUG_ON(list_empty(&root->subsys_list));
    test_ss = list_entry(root->subsys_list.next,
        struct cgroup_subsys, sibling);
    if (css) {
- *css = cont->subsys[test_ss->subsys_id];
+ *css = cgrp->subsys[test_ss->subsys_id];
        BUG_ON(!*css);
    }
    if (subsys_id)
@@ -1157,31 +1157,31 @@ static void get_first_subsys(const struc
}

/*

```

```

- * Attach task 'tsk' to cgroup 'cont'
+ * Attach task 'tsk' to cgroup 'cgrp'
  *
  * Call holding cgroup_mutex. May take task_lock of
  * the task 'pid' during call.
  */
-static int attach_task(struct cgroup *cont, struct task_struct *tsk)
+static int attach_task(struct cgroup *cgrp, struct task_struct *tsk)
{
    int retval = 0;
    struct cgroup_subsys *ss;
- struct cgroup *oldcont;
+ struct cgroup *oldcgrp;
    struct css_set *cg = tsk->cgroups;
    struct css_set *newcg;
- struct cgroupfs_root *root = cont->root;
+ struct cgroupfs_root *root = cgrp->root;
    int subsys_id;

- get_first_subsys(cont, NULL, &subsys_id);
+ get_first_subsys(cgrp, NULL, &subsys_id);

    /* Nothing to do if the task is already in that cgroup */
- oldcont = task_cgroup(tsk, subsys_id);
- if (cont == oldcont)
+ oldcgrp = task_cgroup(tsk, subsys_id);
+ if (cgrp == oldcgrp)
    return 0;

    for_each_subsys(root, ss) {
        if (ss->can_attach) {
-         retval = ss->can_attach(ss, cont, tsk);
+         retval = ss->can_attach(ss, cgrp, tsk);
            if (retval) {
                return retval;
            }
        }
@@ -1192,7 +1192,7 @@ static int attach_task(struct cgroup *co
    * Locate or allocate a new css_set for this task,
    * based on its final set of cgroups
    */
- newcg = find_css_set(cg, cont);
+ newcg = find_css_set(cg, cgrp);
    if (!newcg) {
        return -ENOMEM;
    }
@@ -1216,20 +1216,20 @@ static int attach_task(struct cgroup *co

    for_each_subsys(root, ss) {

```

```

    if (ss->attach) {
- ss->attach(ss, cont, oldcont, tsk);
+ ss->attach(ss, cgrp, oldcgrp, tsk);
    }
}
- set_bit(CONT_RELEASABLE, &oldcont->flags);
+ set_bit(CGRP_RELEASABLE, &oldcgrp->flags);
  synchronize_rcu();
  put_css_set(cg);
  return 0;
}

/*
- * Attach task with pid 'pid' to cgroup 'cont'. Call with
+ * Attach task with pid 'pid' to cgroup 'cgrp'. Call with
  * cgroup_mutex, may take task_lock of task
  */
-static int attach_task_by_pid(struct cgroup *cont, char *pidbuf)
+static int attach_task_by_pid(struct cgroup *cgrp, char *pidbuf)
{
  pid_t pid;
  struct task_struct *tsk;
@@ -1258,7 +1258,7 @@ static int attach_task_by_pid(struct cgr
  get_task_struct(tsk);
}

- ret = attach_task(cont, tsk);
+ ret = attach_task(cgrp, tsk);
  put_task_struct(tsk);
  return ret;
}
@@ -1274,7 +1274,7 @@ enum cgroup_filetype {
  FILE_RELEASE_AGENT,
};

-static ssize_t cgroup_write_uint(struct cgroup *cont, struct cftype *cft,
+static ssize_t cgroup_write_uint(struct cgroup *cgrp, struct cftype *cft,
  struct file *file,
  const char __user *userbuf,
  size_t nbytes, loff_t *unused_ppos)
@@ -1301,13 +1301,13 @@ static ssize_t cgroup_write_uint(struct
  return -EINVAL;

  /* Pass to subsystem */
- retval = cft->write_uint(cont, cft, val);
+ retval = cft->write_uint(cgrp, cft, val);
  if (!retval)
    retval = nbytes;

```

```

return retval;
}

-static ssize_t cgroup_common_file_write(struct cgroup *cont,
+static ssize_t cgroup_common_file_write(struct cgroup *cgrp,
      struct cftype *cft,
      struct file *file,
      const char __user *userbuf,
@@ -1333,25 +1333,25 @@ static ssize_t cgroup_common_file_write(

mutex_lock(&cgroup_mutex);

- if (cgroup_is_removed(cont)) {
+ if (cgroup_is_removed(cgrp)) {
    retval = -ENODEV;
    goto out2;
}

switch (type) {
case FILE_TASKLIST:
- retval = attach_task_by_pid(cont, buffer);
+ retval = attach_task_by_pid(cgrp, buffer);
    break;
case FILE_NOTIFY_ON_RELEASE:
- clear_bit(CONT_RELEASABLE, &cont->flags);
+ clear_bit(CGRP_RELEASABLE, &cgrp->flags);
    if (simple_strtoul(buffer, NULL, 10) != 0)
- set_bit(CONT_NOTIFY_ON_RELEASE, &cont->flags);
+ set_bit(CGRP_NOTIFY_ON_RELEASE, &cgrp->flags);
    else
- clear_bit(CONT_NOTIFY_ON_RELEASE, &cont->flags);
+ clear_bit(CGRP_NOTIFY_ON_RELEASE, &cgrp->flags);
    break;
case FILE_RELEASE_AGENT:
{
- struct cgroupfs_root *root = cont->root;
+ struct cgroupfs_root *root = cgrp->root;
    /* Strip trailing newline */
    if (nbytes && (buffer[nbytes-1] == '\n')) {
        buffer[nbytes-1] = 0;
@@ -1386,30 +1386,30 @@ static ssize_t cgroup_file_write(struct
    size_t nbytes, loff_t *ppos)
{
    struct cftype *cft = __d_cft(file->f_dentry);
- struct cgroup *cont = __d_cont(file->f_dentry->d_parent);
+ struct cgroup *cgrp = __d_cgrp(file->f_dentry->d_parent);

    if (!cft)

```

```

    return -ENODEV;
    if (cft->write)
- return cft->write(cont, cft, file, buf, nbytes, ppos);
+ return cft->write(cgrp, cft, file, buf, nbytes, ppos);
    if (cft->write_uint)
- return cgroup_write_uint(cont, cft, file, buf, nbytes, ppos);
+ return cgroup_write_uint(cgrp, cft, file, buf, nbytes, ppos);
    return -EINVAL;
}

-static ssize_t cgroup_read_uint(struct cgroup *cont, struct cftype *cft,
+static ssize_t cgroup_read_uint(struct cgroup *cgrp, struct cftype *cft,
    struct file *file,
    char __user *buf, size_t nbytes,
    loff_t *ppos)
{
    char tmp[64];
- u64 val = cft->read_uint(cont, cft);
+ u64 val = cft->read_uint(cgrp, cft);
    int len = sprintf(tmp, "%llu\n", (unsigned long long) val);

    return simple_read_from_buffer(buf, nbytes, ppos, tmp, len);
}

-static ssize_t cgroup_common_file_read(struct cgroup *cont,
+static ssize_t cgroup_common_file_read(struct cgroup *cgrp,
    struct cftype *cft,
    struct file *file,
    char __user *buf,
@@ -1431,7 +1431,7 @@ static ssize_t cgroup_common_file_read(s
    struct cgroupfs_root *root;
    size_t n;
    mutex_lock(&cgroup_mutex);
- root = cont->root;
+ root = cgrp->root;
    n = strlen(root->release_agent_path,
        sizeof(root->release_agent_path));
    n = min(n, (size_t) PAGE_SIZE);
@@ -1456,15 +1456,15 @@ static ssize_t cgroup_file_read(struct f
    size_t nbytes, loff_t *ppos)
{
    struct cftype *cft = __d_cft(file->f_dentry);
- struct cgroup *cont = __d_cont(file->f_dentry->d_parent);
+ struct cgroup *cgrp = __d_cgrp(file->f_dentry->d_parent);

    if (!cft)
        return -ENODEV;

```

```

    if (cft->read)
-   return cft->read(cont, cft, file, buf, nbytes, ppos);
+   return cft->read(cgrp, cft, file, buf, nbytes, ppos);
    if (cft->read_uint)
-   return cgroup_read_uint(cont, cft, file, buf, nbytes, ppos);
+   return cgroup_read_uint(cgrp, cft, file, buf, nbytes, ppos);
    return -EINVAL;
}

```

```

@@ -1566,24 +1566,24 @@ static int cgroup_create_file(struct den

```

```

/*
 * cgroup_create_dir - create a directory for an object.
- * cont: the cgroup we create the directory for.
+ * cgrp: the cgroup we create the directory for.
 * It must have a valid ->parent field
 * And we are going to fill its ->dentry field.
- * dentry: dentry of the new container
+ * dentry: dentry of the new cgroup
 * mode: mode to set on new directory.
 */
-static int cgroup_create_dir(struct cgroup *cont, struct dentry *dentry,
+static int cgroup_create_dir(struct cgroup *cgrp, struct dentry *dentry,
    int mode)
{
    struct dentry *parent;
    int error = 0;

-   parent = cont->parent->dentry;
-   error = cgroup_create_file(dentry, S_IFDIR | mode, cont->root->sb);
+   parent = cgrp->parent->dentry;
+   error = cgroup_create_file(dentry, S_IFDIR | mode, cgrp->root->sb);
    if (!error) {
-   dentry->d_fsdata = cont;
+   dentry->d_fsdata = cgrp;
        inc_nlink(parent->d_inode);
-   cont->dentry = dentry;
+   cgrp->dentry = dentry;
        dget(dentry);
    }
    dput(dentry);
@@ -1591,16 +1591,16 @@ static int cgroup_create_dir(struct cgr
    return error;
}

```

```

-int cgroup_add_file(struct cgroup *cont,
+int cgroup_add_file(struct cgroup *cgrp,
    struct cgroup_subsys *subsys,

```



```

    const struct cftype *cft)
{
- struct dentry *dir = cont->dentry;
+ struct dentry *dir = cgrp->dentry;
    struct dentry *dentry;
    int error;

    char name[MAX_CGROUP_TYPE_NAMELEN + MAX_CFTYPE_NAME + 2] = { 0 };
- if (subsys && !test_bit(ROOT_NOPREFIX, &cont->root->flags)) {
+ if (subsys && !test_bit(ROOT_NOPREFIX, &cgrp->root->flags)) {
    strcpy(name, subsys->name);
    strcat(name, ".");
}
@@ -1609,7 +1609,7 @@ int cgroup_add_file(struct cgroup *cont,
    dentry = lookup_one_len(name, dir, strlen(name));
    if (!IS_ERR(dentry)) {
        error = cgroup_create_file(dentry, 0644 | S_IFREG,
-     cont->root->sb);
+     cgrp->root->sb);
        if (!error)
            dentry->d_fsdata = (void *)cft;
        dput(dentry);
@@ -1618,14 +1618,14 @@ int cgroup_add_file(struct cgroup *cont,
    return error;
}

-int cgroup_add_files(struct cgroup *cont,
+int cgroup_add_files(struct cgroup *cgrp,
    struct cgroup_subsys *subsys,
    const struct cftype cft[],
    int count)
{
    int i, err;
    for (i = 0; i < count; i++) {
- err = cgroup_add_file(cont, subsys, &cft[i]);
+ err = cgroup_add_file(cgrp, subsys, &cft[i]);
        if (err)
            return err;
    }
@@ -1634,16 +1634,16 @@ int cgroup_add_files(struct cgroup *cont

/* Count the number of tasks in a cgroup. */

-int cgroup_task_count(const struct cgroup *cont)
+int cgroup_task_count(const struct cgroup *cgrp)
{
    int count = 0;
    struct list_head *l;

```

```

    read_lock(&css_set_lock);
- l = cont->css_sets.next;
- while (l != &cont->css_sets) {
+ l = cgrp->css_sets.next;
+ while (l != &cgrp->css_sets) {
    struct cg_cgroup_link *link =
- list_entry(l, struct cg_cgroup_link, cont_link_list);
+ list_entry(l, struct cg_cgroup_link, cgrp_link_list);
    count += atomic_read(&link->cg->ref.refcount);
    l = l->next;
}
@@ -1655,7 +1655,7 @@ int cgroup_task_count(const struct cgrou
 * Advance a list_head iterator. The iterator should be positioned at
 * the start of a css_set
 */
-static void cgroup_advance_iter(struct cgroup *cont,
+static void cgroup_advance_iter(struct cgroup *cgrp,
    struct cgroup_iter *it)
{
    struct list_head *l = it->cg_link;
@@ -1665,18 +1665,18 @@ static void cgroup_advance_iter(struct c
 /* Advance to the next non-empty css_set */
do {
    l = l->next;
- if (l == &cont->css_sets) {
+ if (l == &cgrp->css_sets) {
    it->cg_link = NULL;
    return;
}
- link = list_entry(l, struct cg_cgroup_link, cont_link_list);
+ link = list_entry(l, struct cg_cgroup_link, cgrp_link_list);
    cg = link->cg;
} while (list_empty(&cg->tasks));
it->cg_link = l;
it->task = cg->tasks.next;
}

-void cgroup_iter_start(struct cgroup *cont, struct cgroup_iter *it)
+void cgroup_iter_start(struct cgroup *cgrp, struct cgroup_iter *it)
{
/*
 * The first time anyone tries to iterate across a cgroup,
@@ -1696,11 +1696,11 @@ void cgroup_iter_start(struct cgroup *co
    write_unlock(&css_set_lock);
}
    read_lock(&css_set_lock);
- it->cg_link = &cont->css_sets;

```

```

- cgroup_advance_iter(cont, it);
+ it->cg_link = &cgrp->css_sets;
+ cgroup_advance_iter(cgrp, it);
}

-struct task_struct *cgroup_iter_next(struct cgroup *cont,
+struct task_struct *cgroup_iter_next(struct cgroup *cgrp,
    struct cgroup_iter *it)
{
    struct task_struct *res;
@@ -1715,14 +1715,14 @@ struct task_struct *cgroup_iter_next(str
    if (l == &res->cgroups->tasks) {
        /* We reached the end of this task list - move on to
         * the next cg_cgroup_link */
- cgroup_advance_iter(cont, it);
+ cgroup_advance_iter(cgrp, it);
    } else {
        it->task = l;
    }
    return res;
}

-void cgroup_iter_end(struct cgroup *cont, struct cgroup_iter *it)
+void cgroup_iter_end(struct cgroup *cgrp, struct cgroup_iter *it)
{
    read_unlock(&css_set_lock);
}
@@ -1748,23 +1748,23 @@ struct ctr_struct {

/*
 * Load into 'pidarray' up to 'npids' of the tasks using cgroup
- * 'cont'. Return actual number of pids loaded. No need to
+ * 'cgrp'. Return actual number of pids loaded. No need to
 * task_lock(p) when reading out p->cgroup, since we're in an RCU
 * read section, so the css_set can't go away, and is
 * immutable after creation.
 */
-static int pid_array_load(pid_t *pidarray, int npids, struct cgroup *cont)
+static int pid_array_load(pid_t *pidarray, int npids, struct cgroup *cgrp)
{
    int n = 0;
    struct cgroup_iter it;
    struct task_struct *tsk;
- cgroup_iter_start(cont, &it);
- while ((tsk = cgroup_iter_next(cont, &it))) {
+ cgroup_iter_start(cgrp, &it);
+ while ((tsk = cgroup_iter_next(cgrp, &it))) {
        if (unlikely(n == npids))

```

```

        break;
        pidarray[n++] = task_pid_nr(tsk);
    }
- cgroup_iter_end(cont, &it);
+ cgroup_iter_end(cgrp, &it);
    return n;
}

@@ -1779,7 +1779,7 @@ static int pid_array_load(pid_t *pidarra
int cgroupstats_build(struct cgroupstats *stats, struct dentry *dentry)
{
    int ret = -EINVAL;
- struct cgroup *cont;
+ struct cgroup *cgrp;
    struct cgroup_iter it;
    struct task_struct *tsk;
    /*
@@ -1789,11 +1789,11 @@ int cgroupstats_build(struct cgroupstats
    goto err;

    ret = 0;
- cont = dentry->d_fsdata;
+ cgrp = dentry->d_fsdata;
    rcu_read_lock();

- cgroup_iter_start(cont, &it);
- while ((tsk = cgroup_iter_next(cont, &it))) {
+ cgroup_iter_start(cgrp, &it);
+ while ((tsk = cgroup_iter_next(cgrp, &it))) {
    switch (tsk->state) {
        case TASK_RUNNING:
            stats->nr_running++;
@@ -1813,7 +1813,7 @@ int cgroupstats_build(struct cgroupstats
        break;
    }
}
- cgroup_iter_end(cont, &it);
+ cgroup_iter_end(cgrp, &it);

    rcu_read_unlock();
err:
@@ -1848,7 +1848,7 @@ static int pid_array_to_buf(char *buf, i
    /*
static int cgroup_tasks_open(struct inode *unused, struct file *file)
{
- struct cgroup *cont = __d_cont(file->f_dentry->d_parent);
+ struct cgroup *cgrp = __d_cgrp(file->f_dentry->d_parent);
    struct ctr_struct *ctr;

```

```

pid_t *pidarray;
int npids;
@@ -1867,13 +1867,13 @@ static int cgroup_tasks_open(struct inode
    * caller from the case that the additional cgroup users didn't
    * show up until sometime later on.
    */
- npids = cgroup_task_count(cont);
+ npids = cgroup_task_count(cgrp);
    if (npids) {
        pidarray = kmalloc(npids * sizeof(pid_t), GFP_KERNEL);
        if (!pidarray)
            goto err1;

- npids = pid_array_load(pidarray, npids, cont);
+ npids = pid_array_load(pidarray, npids, cgrp);
        sort(pidarray, npids, sizeof(pid_t), cmp_pid, NULL);

        /* Call pid_array_to_buf() twice, first just to get bufsz */
@@ -1899,7 +1899,7 @@ err0:
        return -ENOMEM;
    }

-static ssize_t cgroup_tasks_read(struct cgroup *cont,
+static ssize_t cgroup_tasks_read(struct cgroup *cgrp,
    struct cftype *cft,
    struct file *file, char __user *buf,
    size_t nbytes, loff_t *ppos)
@@ -1922,15 +1922,15 @@ static int cgroup_tasks_release(struct i
    return 0;
}

-static u64 cgroup_read_notify_on_release(struct cgroup *cont,
+static u64 cgroup_read_notify_on_release(struct cgroup *cgrp,
    struct cftype *cft)
{
- return notify_on_release(cont);
+ return notify_on_release(cgrp);
}

-static u64 cgroup_read_releasable(struct cgroup *cont, struct cftype *cft)
+static u64 cgroup_read_releasable(struct cgroup *cgrp, struct cftype *cft)
{
- return test_bit(CONT_RELEASABLE, &cont->flags);
+ return test_bit(CGRP_RELEASABLE, &cgrp->flags);
}

/*
@@ -1967,25 +1967,25 @@ static struct cftype cft_release_agent =

```

```

.private = FILE_RELEASE_AGENT,
};

-static int cgroup_populate_dir(struct cgroup *cont)
+static int cgroup_populate_dir(struct cgroup *cgrp)
{
    int err;
    struct cgroup_subsys *ss;

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

- err = cgroup_add_files(cont, NULL, files, ARRAY_SIZE(files));
+ err = cgroup_add_files(cgrp, NULL, files, ARRAY_SIZE(files));
    if (err < 0)
        return err;

- if (cont == cont->top_cgroup) {
-     if ((err = cgroup_add_file(cont, NULL, &cft_release_agent)) < 0)
+ if (cgrp == cgrp->top_cgroup) {
+     if ((err = cgroup_add_file(cgrp, NULL, &cft_release_agent)) < 0)
        return err;
    }

- for_each_subsys(cont->root, ss) {
-     if (ss->populate && (err = ss->populate(ss, cont)) < 0)
+ for_each_subsys(cgrp->root, ss) {
+     if (ss->populate && (err = ss->populate(ss, cgrp)) < 0)
        return err;
    }

@@ -1994,15 +1994,15 @@ static int cgroup_populate_dir(struct cg

static void init_cgroup_css(struct cgroup_subsys_state *css,
                           struct cgroup_subsys *ss,
-                           struct cgroup *cont)
+                           struct cgroup *cgrp)
{
- css->cgroup = cont;
+ css->cgroup = cgrp;
    atomic_set(&css->refcnt, 0);
    css->flags = 0;
- if (cont == dummytop)
+ if (cgrp == dummytop)
    set_bit(CSS_ROOT, &css->flags);
- BUG_ON(cont->subsys[ss->subsys_id]);
- cont->subsys[ss->subsys_id] = css;

```

```

+ BUG_ON(cgrp->subsys[ss->subsys_id]);
+ cgrp->subsys[ss->subsys_id] = css;
}

/*
@@ -2017,14 +2017,14 @@ static void init_cgroup_css(struct cgrou
static long cgroup_create(struct cgroup *parent, struct dentry *dentry,
int mode)
{
- struct cgroup *cont;
+ struct cgroup *cgrp;
  struct cgroupfs_root *root = parent->root;
  int err = 0;
  struct cgroup_subsys *ss;
  struct super_block *sb = root->sb;

- cont = kzalloc(sizeof(*cont), GFP_KERNEL);
- if (!cont)
+ cgrp = kzalloc(sizeof(*cgrp), GFP_KERNEL);
+ if (!cgrp)
  return -ENOMEM;

  /* Grab a reference on the superblock so the hierarchy doesn't
@@ -2036,53 +2036,53 @@ static long cgroup_create(struct cgroup

  mutex_lock(&cgroup_mutex);

- cont->flags = 0;
- INIT_LIST_HEAD(&cont->sibling);
- INIT_LIST_HEAD(&cont->children);
- INIT_LIST_HEAD(&cont->css_sets);
- INIT_LIST_HEAD(&cont->release_list);
-
- cont->parent = parent;
- cont->root = parent->root;
- cont->top_cgroup = parent->top_cgroup;
+ cgrp->flags = 0;
+ INIT_LIST_HEAD(&cgrp->sibling);
+ INIT_LIST_HEAD(&cgrp->children);
+ INIT_LIST_HEAD(&cgrp->css_sets);
+ INIT_LIST_HEAD(&cgrp->release_list);
+
+ cgrp->parent = parent;
+ cgrp->root = parent->root;
+ cgrp->top_cgroup = parent->top_cgroup;

  for_each_subsys(root, ss) {
- struct cgroup_subsys_state *css = ss->create(ss, cont);

```

```

+ struct cgroup_subsys_state *css = ss->create(ss, cgrp);
  if (IS_ERR(css)) {
    err = PTR_ERR(css);
    goto err_destroy;
  }
- init_cgroup_css(css, ss, cont);
+ init_cgroup_css(css, ss, cgrp);
}

- list_add(&cont->sibling, &cont->parent->children);
+ list_add(&cgrp->sibling, &cgrp->parent->children);
  root->number_of_cgroups++;

- err = cgroup_create_dir(cont, dentry, mode);
+ err = cgroup_create_dir(cgrp, dentry, mode);
  if (err < 0)
    goto err_remove;

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

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

  mutex_unlock(&cgroup_mutex);
- mutex_unlock(&cont->dentry->d_inode->i_mutex);
+ mutex_unlock(&cgrp->dentry->d_inode->i_mutex);

  return 0;

err_remove:

- list_del(&cont->sibling);
+ list_del(&cgrp->sibling);
  root->number_of_cgroups--;

err_destroy:

  for_each_subsys(root, ss) {
-   if (cont->subsys[ss->subsys_id])
-     ss->destroy(ss, cont);
+   if (cgrp->subsys[ss->subsys_id])
+     ss->destroy(ss, cgrp);
  }

  mutex_unlock(&cgroup_mutex);

```



```

@@ -2090,7 +2090,7 @@ static long cgroup_create(struct cgroup
/* Release the reference count that we took on the superblock */
deactivate_super(sb);

- kfree(cont);
+ kfree(cgrp);
return err;
}

@@ -2102,7 +2102,7 @@ static int cgroup_mkdir(struct inode *di
return cgroup_create(c_parent, dentry, mode | S_IFDIR);
}

-static inline int cgroup_has_css_refs(struct cgroup *cont)
+static inline int cgroup_has_css_refs(struct cgroup *cgrp)
{
/* Check the reference count on each subsystem. Since we
* already established that there are no tasks in the
@@ -2118,9 +2118,9 @@ static inline int cgroup_has_css_refs(st
struct cgroup_subsys *ss = subsys[i];
struct cgroup_subsys_state *css;
/* Skip subsystems not in this hierarchy */
- if (ss->root != cont->root)
+ if (ss->root != cgrp->root)
continue;
- css = cont->subsys[ss->subsys_id];
+ css = cgrp->subsys[ss->subsys_id];
/* When called from check_for_release() it's possible
* that by this point the cgroup has been removed
* and the css deleted. But a false-positive doesn't
@@ -2136,7 +2136,7 @@ static inline int cgroup_has_css_refs(st

static int cgroup_rmdir(struct inode *unused_dir, struct dentry *dentry)
{
- struct cgroup *cont = dentry->d_fsdata;
+ struct cgroup *cgrp = dentry->d_fsdata;
struct dentry *d;
struct cgroup *parent;
struct cgroup_subsys *ss;
@@ -2146,46 +2146,46 @@ static int cgroup_rmdir(struct inode *un
/* the vfs holds both inode->i_mutex already */

mutex_lock(&cgroup_mutex);
- if (atomic_read(&cont->count) != 0) {
+ if (atomic_read(&cgrp->count) != 0) {
mutex_unlock(&cgroup_mutex);
return -EBUSY;
}

```

```

- if (!list_empty(&cont->children)) {
+ if (!list_empty(&cgrp->children)) {
    mutex_unlock(&cgroup_mutex);
    return -EBUSY;
}

- parent = cont->parent;
- root = cont->root;
+ parent = cgrp->parent;
+ root = cgrp->root;
    sb = root->sb;

- if (cgroup_has_css_refs(cont)) {
+ if (cgroup_has_css_refs(cgrp)) {
    mutex_unlock(&cgroup_mutex);
    return -EBUSY;
}

    for_each_subsys(root, ss) {
- if (cont->subsys[ss->subsys_id])
- ss->destroy(ss, cont);
+ if (cgrp->subsys[ss->subsys_id])
+ ss->destroy(ss, cgrp);
    }

    spin_lock(&release_list_lock);
- set_bit(CONT_REMOVED, &cont->flags);
- if (!list_empty(&cont->release_list))
- list_del(&cont->release_list);
+ set_bit(CGRP_REMOVED, &cgrp->flags);
+ if (!list_empty(&cgrp->release_list))
+ list_del(&cgrp->release_list);
    spin_unlock(&release_list_lock);
    /* delete my sibling from parent->children */
- list_del(&cont->sibling);
- spin_lock(&cont->dentry->d_lock);
- d = dget(cont->dentry);
- cont->dentry = NULL;
+ list_del(&cgrp->sibling);
+ spin_lock(&cgrp->dentry->d_lock);
+ d = dget(cgrp->dentry);
+ cgrp->dentry = NULL;
    spin_unlock(&d->d_lock);

    cgroup_d_remove_dir(d);
    dput(d);
    root->number_of_cgroups--;

```

```

- set_bit(CONT_RELEASABLE, &parent->flags);
+ set_bit(CGRP_RELEASABLE, &parent->flags);
  check_for_release(parent);

  mutex_unlock(&cgroup_mutex);
@@ -2259,7 +2259,7 @@ int __init cgroup_init_early(void)
  init_task.cgroups = &init_css_set;

  init_css_set_link.cg = &init_css_set;
- list_add(&init_css_set_link.cont_link_list,
+ list_add(&init_css_set_link.cgrp_link_list,
  &rootnode.top_cgroup.css_sets);
  list_add(&init_css_set_link.cg_link_list,
  &init_css_set.cg_links);
@@ -2356,7 +2356,7 @@ static int proc_cgroup_show(struct seq_f

  for_each_root(root) {
    struct cgroup_subsys *ss;
-   struct cgroup *cont;
+   struct cgroup *cgrp;
    int subsys_id;
    int count = 0;

@@ -2367,8 +2367,8 @@ static int proc_cgroup_show(struct seq_f
    seq_printf(m, "%s%s", count++ ? " : " : "", ss->name);
    seq_putc(m, ':');
    get_first_subsys(&root->top_cgroup, NULL, &subsys_id);
-   cont = task_cgroup(tsk, subsys_id);
-   retval = cgroup_path(cont, buf, PAGE_SIZE);
+   cgrp = task_cgroup(tsk, subsys_id);
+   retval = cgroup_path(cgrp, buf, PAGE_SIZE);
    if (retval < 0)
      goto out_unlock;
    seq_puts(m, buf);
@@ -2630,7 +2630,7 @@ int cgroup_clone(struct task_struct *tsk

  /* Create the cgroup directory, which also creates the cgroup */
  ret = vfs_mkdir(inode, dentry, S_IFDIR | 0755);
- child = __d_cont(dentry);
+ child = __d_cgrp(dentry);
  dput(dentry);
  if (ret) {
    printk(KERN_INFO
@@ -2687,7 +2687,7 @@ int cgroup_clone(struct task_struct *tsk
  }

  /*
- * See if "cont" is a descendant of the current task's cgroup in

```

```

+ * See if "cgrp" is a descendant of the current task's cgroup in
  * the appropriate hierarchy
  *
  * If we are sending in dummytop, then presumably we are creating
@@ -2695,37 +2695,37 @@ int cgroup_clone(struct task_struct *tsk
  *
  * Called only by the ns (nsproxy) cgroup.
  */
-int cgroup_is_descendant(const struct cgroup *cont)
+int cgroup_is_descendant(const struct cgroup *cgrp)
{
    int ret;
    struct cgroup *target;
    int subsys_id;

- if (cont == dummytop)
+ if (cgrp == dummytop)
    return 1;

- get_first_subsys(cont, NULL, &subsys_id);
+ get_first_subsys(cgrp, NULL, &subsys_id);
    target = task_cgroup(current, subsys_id);
- while (cont != target && cont != cont->top_cgroup)
-     cont = cont->parent;
- ret = (cont == target);
+ while (cgrp != target && cgrp != cgrp->top_cgroup)
+     cgrp = cgrp->parent;
+ ret = (cgrp == target);
    return ret;
}

-static void check_for_release(struct cgroup *cont)
+static void check_for_release(struct cgroup *cgrp)
{
    /* All of these checks rely on RCU to keep the cgroup
     * structure alive */
- if (cgroup_is_releasable(cont) && !atomic_read(&cont->count)
-     && list_empty(&cont->children) && !cgroup_has_css_refs(cont)) {
+ if (cgroup_is_releasable(cgrp) && !atomic_read(&cgrp->count)
+     && list_empty(&cgrp->children) && !cgroup_has_css_refs(cgrp)) {
    /* Control Group is currently removeable. If it's not
     * already queued for a userspace notification, queue
     * it now */
    int need_schedule_work = 0;
    spin_lock(&release_list_lock);
- if (!cgroup_is_removed(cont) &&
-     list_empty(&cont->release_list)) {
-     list_add(&cont->release_list, &release_list);

```

```

+ if (!cgroup_is_removed(cgrp) &&
+   list_empty(&cgrp->release_list)) {
+   list_add(&cgrp->release_list, &release_list);
+   need_schedule_work = 1;
+ }
+ spin_unlock(&release_list_lock);
@@ -2736,11 +2736,11 @@ static void check_for_release(struct cgr

void __css_put(struct cgroup_subsys_state *css)
{
- struct cgroup *cont = css->cgroup;
+ struct cgroup *cgrp = css->cgroup;
+   rcu_read_lock();
- if (atomic_dec_and_test(&css->refcnt) && notify_on_release(cont)) {
-   set_bit(CONT_RELEASABLE, &cont->flags);
-   check_for_release(cont);
+ if (atomic_dec_and_test(&css->refcnt) && notify_on_release(cgrp)) {
+   set_bit(CGRP_RELEASABLE, &cgrp->flags);
+   check_for_release(cgrp);
+ }
+   rcu_read_unlock();
+ }
@@ -2779,10 +2779,10 @@ static void cgroup_release_agent(struct
char *argv[3], *envp[3];
int i;
char *pathbuf;
- struct cgroup *cont = list_entry(release_list.next,
+ struct cgroup *cgrp = list_entry(release_list.next,
+   struct cgroup,
+   release_list);
- list_del_init(&cont->release_list);
+ list_del_init(&cgrp->release_list);
+   spin_unlock(&release_list_lock);
+   pathbuf = kmalloc(PAGE_SIZE, GFP_KERNEL);
+   if (!pathbuf) {
@@ -2790,14 +2790,14 @@ static void cgroup_release_agent(struct
continue;
+ }

- if (cgroup_path(cont, pathbuf, PAGE_SIZE) < 0) {
+ if (cgroup_path(cgrp, pathbuf, PAGE_SIZE) < 0) {
+   kfree(pathbuf);
+   spin_lock(&release_list_lock);
+   continue;
+ }

+ i = 0;
- argv[i++] = cont->root->release_agent_path;

```

```
+ argv[i++] = cgrp->root->release_agent_path;  
  argv[i++] = (char *)pathbuf;  
  argv[i] = NULL;
```

Containers mailing list

Containers@lists.linux-foundation.org

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