Subject: Re: [PATCH 2/7] cgroup: fix comments
Posted by Li Zefan on Thu, 21 Feb 2008 02:14:49 GMT
View Forum Message <> Reply to Message

Paul Menage wrote:
> I think that docbook-style function comments need /** at the start of
> the comment block.
>

Yes, I didn't notice it. I revised the patch to fix it.

---

fix:
- comments about need_forkexit_callback
- comments about release agent
- typo and comment style, etc.

Signed-off-by: Li Zefan <lizf@cn.fujitsu.com>

```
---
 include/linux/cgroup.h |    2 +-
 kernel/cgroup.c        |  142 ++++++++++++++++++++++++++++---------------------
 2 files changed, 80 insertions(+), 64 deletions(-)

diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
index ff9055f..2ebf7af 100644
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -175,7 +175,7 @@ struct css_set {
  *
  *
  * When reading/writing to a file:
- * - the cgroup to use in file->f_dentry->d_parent->d_fsdata
+ * - the cgroup to use is file->f_dentry->d_parent->d_fsdata
  * - the 'cftype' of the file is file->f_dentry->d_fsdata
  */

diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index 4766bb6..36066d8 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -113,9 +113,9 @@ static int root_count;
 #define dummytop (&rootnode.top_cgroup)

 /* This flag indicates whether tasks in the fork and exit paths should
- * take callback_mutex and check for fork/exit handlers to call. This
- * avoids us having to do extra work in the fork/exit path if none of the
- * subsystems need to be called.
+ * check for fork/exit handlers to call. This avoids us having to do
+ * extra work in the fork/exit path if none of the subsystems need to
+ * be called.
  */
 static int need_forkexit_callback;

@@ -307,7 +307,6 @@ static inline void put_css_set_taskexit(struct css_set *cg)
  * template: location in which to build the desired set of subsystem
  * state objects for the new cgroup group
  */
-
 static struct css_set *find_existing_css_set(
  struct css_set *oldcg,
  struct cgroup *cgrp,
@@ -354,7 +353,6 @@ static struct css_set *find_existing_css_set(
  * and chains them on tmp through their cgrp_link_list fields. Returns 0 on
  * success or a negative error
  */
-
 static int allocate_cg_links(int count, struct list_head *tmp)
 {
  struct cg_cgroup_link *link;
@@ -396,7 +394,6 @@ static void free_cg_links(struct list_head *tmp)
  * substituted into the appropriate hierarchy. Must be called with
  * cgroup_mutex held
  */
-
 static struct css_set *find_css_set(
  struct css_set *oldcg, struct cgroup *cgrp)
 {
@@ -507,8 +504,8 @@ static struct css_set *find_css_set(
  * critical pieces of code here.  The exception occurs on cgroup_exit(),
  * when a task in a notify_on_release cgroup exits.  Then cgroup_mutex
  * is taken, and if the cgroup count is zero, a usermode call made
- * to /sbin/cgroup_release_agent with the name of the cgroup (path
- * relative to the root of cgroup file system) as the argument.
+ * to the release agent with the name of the cgroup (path relative to
+ * the root of cgroup file system) as the argument.
  *
  * A cgroup can only be deleted if both its 'count' of using tasks
  * is zero, and its list of 'children' cgroups is empty.  Since all
@@ -521,7 +518,7 @@ static struct css_set *find_css_set(
  *
  * The need for this exception arises from the action of
  * cgroup_attach_task(), which overwrites one tasks cgroup pointer with
- * another.  It does so using cgroup_mutexe, however there are
+ * another.  It does so using cgroup_mutex, however there are
```

```
  * several performance critical places that need to reference
  * task->cgroup without the expense of grabbing a system global
  * mutex.  Therefore except as noted below, when dereferencing or, as
@@ -537,7 +534,6 @@ static struct css_set *find_css_set(
  * cgroup_lock - lock out any changes to cgroup structures
  *
  */
-
 void cgroup_lock(void)
 {
  mutex_lock(&cgroup_mutex);
@@ -548,7 +544,6 @@ void cgroup_lock(void)
  *
  * Undo the lock taken in a previous cgroup_lock() call.
  */
-
 void cgroup_unlock(void)
 {
  mutex_unlock(&cgroup_mutex);
@@ -590,7 +585,6 @@ static struct inode *cgroup_new_inode(mode_t mode, struct super_block
*sb)
  * Call subsys's pre_destroy handler.
  * This is called before css refcnt check.
  */
-
 static void cgroup_call_pre_destroy(struct cgroup *cgrp)
 {
  struct cgroup_subsys *ss;
@@ -600,7 +594,6 @@ static void cgroup_call_pre_destroy(struct cgroup *cgrp)
  return;
 }

-
 static void cgroup_diput(struct dentry *dentry, struct inode *inode)
 {
  /* is dentry a directory ? if so, kfree() associated cgroup */
@@ -1129,8 +1122,13 @@ static inline struct cftype *__d_cft(struct dentry *dentry)
  return dentry->d_fsdata;
 }

-/*
- * Called with cgroup_mutex held.  Writes path of cgroup into buf.
+/**
+ * cgroup_path - generate the path of a cgroup
+ * @cgrp: the cgroup in question
+ * @buf: the buffer to write the path into
+ * @buflen: the length of the buffer
+ *
+ * Called with cgroup_mutex held. Writes path of cgroup into buf.
  * Returns 0 on success, -errno on error.
  */
 int cgroup_path(const struct cgroup *cgrp, char *buf, int buflen)
@@ -1188,11 +1186,13 @@ static void get_first_subsys(const struct cgroup *cgrp,
   *subsys_id = test_ss->subsys_id;
 }

-/*
- * Attach task 'tsk' to cgroup 'cgrp'
+/**
+ * cgroup_attach_task - attach task 'tsk' to cgroup 'cgrp'
+ * @cgrp: the cgroup the task is attaching to
+ * @tsk: the task to be attached
  *
- * Call holding cgroup_mutex.  May take task_lock of
- * the task 'pid' during call.
+ * Call holding cgroup_mutex. May take task_lock of
+ * the task 'tsk' during call.
  */
 int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
 {
@@ -1293,7 +1293,6 @@ static int attach_task_by_pid(struct cgroup *cgrp, char *pidbuf)
 }

 /* The various types of files and directories in a cgroup file system */
-
 enum cgroup_filetype {
  FILE_ROOT,
  FILE_DIR,
@@ -1584,12 +1583,11 @@ static int cgroup_create_file(struct dentry *dentry, int mode,
 }

 /*
- * cgroup_create_dir - create a directory for an object.
- * 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 cgroup
- * mode: mode to set on new directory.
+ * cgroup_create_dir - create a directory for an object.
+ * @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 cgroup
+ * @mode: mode to set on new directory.
  */
 static int cgroup_create_dir(struct cgroup *cgrp, struct dentry *dentry,
     int mode)
```

```
@@ -1651,8 +1649,12 @@ int cgroup_add_files(struct cgroup *cgrp,
  return 0;
 }

-/* Count the number of tasks in a cgroup. */
-
+/**
+ * cgroup_task_count - count the number of tasks in a cgroup.
+ * @cgrp: the cgroup in question
+ *
+ * Return the number of tasks in the cgroup.
+ */
 int cgroup_task_count(const struct cgroup *cgrp)
 {
  int count = 0;
@@ -1962,12 +1964,13 @@ static int pid_array_load(pid_t *pidarray, int npids, struct cgroup
*cgrp)
 }

 /**
- * Build and fill cgroupstats so that taskstats can export it to user
- * space.
- *
+ * cgroupstats_build - build and fill cgroupstats
  * @stats: cgroupstats to fill information into
  * @dentry: A dentry entry belonging to the cgroup for which stats have
  * been requested.
+ *
+ * Build and fill cgroupstats so that taskstats can export it to user
+ * space.
  */
 int cgroupstats_build(struct cgroupstats *stats, struct dentry *dentry)
 {
@@ -2199,14 +2202,13 @@ static void init_cgroup_css(struct cgroup_subsys_state *css,
 }

 /*
- * cgroup_create - create a cgroup
- * parent: cgroup that will be parent of the new cgroup.
- * name:  name of the new cgroup. Will be strcpy'ed.
- * mode:  mode to set on new inode
+ * cgroup_create - create a cgroup
+ * @parent: cgroup that will be parent of the new cgroup
+ * @dentry: dentry of the new cgroup
+ * @mode: mode to set on new inode
  *
- * Must be called with the mutex on the parent inode held
+ * Must be called with the mutex on the parent inode held
```

```
  */
-
 static long cgroup_create(struct cgroup *parent, struct dentry *dentry,
         int mode)
 {
@@ -2349,13 +2351,12 @@ static int cgroup_rmdir(struct inode *unused_dir, struct dentry
*dentry)
  parent = cgrp->parent;
  root = cgrp->root;
  sb = root->sb;
+
  /*
-  * Call pre_destroy handlers of subsys
+  * Call pre_destroy handlers of subsys. Notify subsystems
+  * that rmdir() request comes.
   */
  cgroup_call_pre_destroy(cgrp);
- /*
-  * Notify subsyses that rmdir() request comes.
-  */

  if (cgroup_has_css_refs(cgrp)) {
   mutex_unlock(&cgroup_mutex);
@@ -2431,8 +2432,10 @@ static void cgroup_init_subsys(struct cgroup_subsys *ss)
 }

 /**
- * cgroup_init_early - initialize cgroups at system boot, and
- * initialize any subsystems that request early init.
+ * cgroup_init_early - cgroup initialization at system boot
+ *
+ * Initialize cgroups at system boot, and initialize any
+ * subsystems that request early init.
  */
 int __init cgroup_init_early(void)
 {
@@ -2474,8 +2477,10 @@ int __init cgroup_init_early(void)
 }

 /**
- * cgroup_init - register cgroup filesystem and /proc file, and
- * initialize any subsystems that didn't request early init.
+ * cgroup_init - cgroup initialization
+ *
+ * Register cgroup filesystem and /proc file, and initialize
+ * any subsystems that didn't request early init.
  */
 int __init cgroup_init(void)
 {
@@ -2618,7 +2623,7 @@ static struct file_operations proc_cgroupstats_operations = {

 /**
  * cgroup_fork - attach newly forked task to its parents cgroup.
- * @tsk: pointer to task_struct of forking parent process.
+ * @child: pointer to task_struct of forking parent process.
  *
  * Description: A task inherits its parent's cgroup at fork().
  *
@@ -2642,9 +2647,12 @@ void cgroup_fork(struct task_struct *child)
 }

 /**
- * cgroup_fork_callbacks - called on a new task very soon before
- * adding it to the tasklist. No need to take any locks since no-one
- * can be operating on this task
+ * cgroup_fork_callbacks - run fork callbacks
+ * @child: the new task
+ *
+ * Called on a new task very soon before adding it to the
+ * tasklist. No need to take any locks since no-one can
+ * be operating on this task.
  */
 void cgroup_fork_callbacks(struct task_struct *child)
 {
@@ -2659,11 +2667,14 @@ void cgroup_fork_callbacks(struct task_struct *child)
 }

 /**
- * cgroup_post_fork - called on a new task after adding it to the
- * task list. Adds the task to the list running through its css_set
- * if necessary. Has to be after the task is visible on the task list
- * in case we race with the first call to cgroup_iter_start() - to
- * guarantee that the new task ends up on its list. */
+ * cgroup_post_fork - called on a new task after adding it to the task list
+ * @child: the task in question
+ *
+ * Adds the task to the list running through its css_set if necessary.
+ * Has to be after the task is visible on the task list in case we race
+ * with the first call to cgroup_iter_start() - to guarantee that the
+ * new task ends up on its list.
+ */
 void cgroup_post_fork(struct task_struct *child)
 {
  if (use_task_css_set_links) {
@@ -2676,6 +2687,7 @@ void cgroup_post_fork(struct task_struct *child)
 /**
```

```
  * cgroup_exit - detach cgroup from exiting task
  * @tsk: pointer to task_struct of exiting process
+ * @run_callback: run exit callbacks?
  *
  * Description: Detach cgroup from @tsk and release it.
  *
@@ -2706,7 +2718,6 @@ void cgroup_post_fork(struct task_struct *child)
  *    top_cgroup isn't going away, and either task has PF_EXITING set,
  *    which wards off any cgroup_attach_task() attempts, or task is a failed
  *    fork, never visible to cgroup_attach_task.
- *
  */
 void cgroup_exit(struct task_struct *tsk, int run_callbacks)
 {
@@ -2743,9 +2754,13 @@ void cgroup_exit(struct task_struct *tsk, int run_callbacks)
 }

 /**
- * cgroup_clone - duplicate the current cgroup in the hierarchy
- * that the given subsystem is attached to, and move this task into
- * the new child
+ * cgroup_clone - clone the cgroup the given subsystem is attached to
+ * @tsk: the task to be moved
+ * @subsys: the given subsystem
+ *
+ * Duplicate the current cgroup in the hierarchy that the given
+ * subsystem is attached to, and move this task into the new
+ * child.
  */
 int cgroup_clone(struct task_struct *tsk, struct cgroup_subsys *subsys)
 {
@@ -2858,9 +2873,12 @@ int cgroup_clone(struct task_struct *tsk, struct cgroup_subsys
*subsys)
  return ret;
 }

-/*
- * See if "cgrp" is a descendant of the current task's cgroup in
- * the appropriate hierarchy
+/**
+ * cgroup_is_descendant - see if @cgrp is a descendant of current task's cgrp
+ * @cgrp: the cgroup in question
+ *
+ * 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
  * the top cgroup in the subsystem.
```

```
@@ -2939,9 +2957,7 @@ void __css_put(struct cgroup_subsys_state *css)
  * release agent task.  We don't bother to wait because the caller of
  * this routine has no use for the exit status of the release agent
  * task, so no sense holding our caller up for that.
- *
  */
-
 static void cgroup_release_agent(struct work_struct *work)
 {
  BUG_ON(work != &release_agent_work);
--
1.5.4.rc3
```

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers