
Subject: Re: [RFC] [PATCH] cgroup: add "procs" control file
Posted by KAMEZAWA Hiroyuki on Thu, 19 Jun 2008 00:14:36 GMT
[View Forum Message](#) <[Reply to Message](#)

On Wed, 18 Jun 2008 16:02:25 +0800

Li Zefan <lizf@cn.fujitsu.com> wrote:

> This control file is the equivalent of the "tasks" control file, but
> acting/reporting on entire thread groups.
>
> For example, we have a process with pid 1000 and its sub-thread with
> tid 1001, to attach them into a cgroup:
> # echo 1000 > procs
> Then show the process list and the task list respectively:
> # cat procs
> 1000
> # cat tasks
> 1000
> 1001
>
> Questions:
> - What to do if the attaching of a thread failed? continue to attach
> other threads, or stop and return error?
> - When a sub-thread of a process is in the cgroup, but not its thread
> cgroup leader, what to do when 'cat procs'? just skip those threads?
>

I think this feature make sense. But not meets a theory that cgroup handles a thread not a process. So, how about changing the definition of this interface from

- showing procs
to
- showing threads which is thread-group-leader.

One possible problem is a case that thread-group-leader exits while other members are alive. In such case, thread-group-leader calls cgroup_exit() but will be still alive until all sub-threads exit. So, this interface cannot show correct information.

(right ??? please point out if I miss something)

So, how about this kind of interface ? showing both of TID and PID.

```
%/cat/procs
TID PID
1001 1001
1234 1001
3856 1001
728 728
....
```

....

nonsense ?

Thanks,
-Kame

```
> Signed-off-by: Li Zefan <lizf@cn.fujitsu.com>
> ---
> Documentation/cgroups.txt |  9 +++
> include/linux/cgroup.h   |  2 ++
> kernel/cgroup.c         | 112 ++++++=====
> kernel/cpuset.c          |  2 ++
> 4 files changed, 107 insertions(+), 18 deletions(-)
>
> diff --git a/Documentation/cgroups.txt b/Documentation/cgroups.txt
> index 8252f5b..03c7b21 100644
> --- a/Documentation/cgroups.txt
> +++ b/Documentation/cgroups.txt
> @@ -227,6 +227,7 @@ Each cgroup is represented by a directory in the cgroup file system
> containing the following files describing that cgroup:
>
> - tasks: list of tasks (by pid) attached to that cgroup
> + - procs: list of processes (by pid) attached to that cgroup
> - releasable flag: cgroup currently removable?
> - notify_on_release flag: run the release agent on exit?
> - release_agent: the path to use for release notifications (this file
> @@ -380,6 +381,8 @@ reference).
> 2.2 Attaching processes
> -----
>
> +To attach a task to a cgroup:
> +
> # /bin/echo PID > tasks
>
> Note that it is PID, not PIDs. You can only attach ONE task at a time.
> @@ -390,6 +393,12 @@ If you have several tasks to attach, you have to do it one after
another:
> ...
> # /bin/echo PIDn > tasks
>
> +To attach an entire thread group to a cgroup:
> +
> +# /bin/echo PID > procs
> +
```

```

> +PID should be the pid of the thread group leader.
> +
> 3. Kernel API
> =====
>
> diff --git a/include/linux/cgroup.h b/include/linux/cgroup.h
> index fc99ba4..f125324 100644
> --- a/include/linux/cgroup.h
> +++ b/include/linux/cgroup.h
> @@ -376,7 +376,7 @@ struct task_struct *cgroup_iter_next(struct cgroup *cgrp,
>     struct cgroup_iter *it);
> void cgroup_iter_end(struct cgroup *cgrp, struct cgroup_iter *it);
> int cgroup_scan_tasks(struct cgroup_scanner *scan);
> -int cgroup_attach_task(struct cgroup *, struct task_struct *);
> +int cgroup_attach_task(struct cgroup *, struct task_struct *, struct css_set *);
>
> #else /* !CONFIG_CGROUPS */
>
> diff --git a/kernel/cgroup.c b/kernel/cgroup.c
> index a4c8671..2fcf733 100644
> --- a/kernel/cgroup.c
> +++ b/kernel/cgroup.c
> @@ -1246,17 +1246,18 @@ static void get_first_subsys(const struct 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
> + * @newcg: the new css_set for this task, can be NULL
> *
> * 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)
> +int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk,
> +    struct css_set *newcg)
> {
>     int retval = 0;
>     struct cgroup_subsys *ss;
>     struct cgroup *oldcgrp;
>     struct css_set *cg = tsk->cgroups;
> -    struct css_set *newcg;
>     struct cgroupfs_root *root = cgrp->root;
>     int subsys_id;
>
> @@ -1279,9 +1280,12 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
>     * Locate or allocate a new css_set for this task,
>     * based on its final set of cgroups
> */
> -    newcg = find_css_set(cg, cgrp);

```

```

> - if (!newcg)
> - return -ENOMEM;
> + if (!newcg) {
> + newcg = find_css_set(cg, cgrp);
> + if (!newcg)
> + return -ENOMEM;
> + } else
> + get_css_set(newcg);
>
> task_lock(tsk);
> if (tsk->flags & PF_EXITING) {
> @@ -1311,24 +1315,23 @@ int cgroup_attach_task(struct cgroup *cgrp, struct task_struct *tsk)
> }
>
> /*
> - * Attach task with pid 'pid' to cgroup 'cgrp'. Call with
> - * cgroup_mutex, may take task_lock of task
> + * Get the task with the provided pid. The caller should
> + * call put_task_struct() with the returned task.
> */
>-static int attach_task_by_pid(struct cgroup *cgrp, char *pidbuf)
>+static struct task_struct *attach_get_task(struct cgroup *cgrp, char *pidbuf)
>{
>    pid_t pid;
>    struct task_struct *tsk;
>    int ret;
>
>    if (sscanf(pidbuf, "%d", &pid) != 1)
> -    return -EIO;
> +    return ERR_PTR(-EIO);
>
>    if (pid) {
>        rcu_read_lock();
>        tsk = find_task_by_vpid(pid);
>        if (!tsk || tsk->flags & PF_EXITING) {
>            rcu_read_unlock();
> -        return -ESRCH;
> +        return ERR_PTR(-ESRCH);
>        }
>        get_task_struct(tsk);
>        rcu_read_unlock();
> @@ -1336,23 +1339,82 @@ static int attach_task_by_pid(struct cgroup *cgrp, char *pidbuf)
>        if ((current->euid) && (current->euid != tsk->uid)
>            && (current->euid != tsk->suid)) {
>            put_task_struct(tsk);
> -        return -EACCES;
> +        return ERR_PTR(-EACCES);
>    }

```

```

> } else {
>   tsk = current;
>   get_task_struct(tsk);
> }
>
> - ret = cgroup_attach_task(cgrp, tsk);
> + return tsk;
> +}
> +
> +/*
> + * 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 *cgrp, char *pidbuf)
> +{
> + struct task_struct *tsk;
> + int ret;
> +
> + tsk = attach_get_task(cgrp, pidbuf);
> + if (IS_ERR(tsk))
> +   return PTR_ERR(tsk);
> +
> + ret = cgroup_attach_task(cgrp, tsk, NULL);
> + put_task_struct(tsk);
> + return ret;
> }
>
> +/*
> + * Attach the entire thread groups whose group leader is 'pid'. Call
> + * with cgroup_mutex, may take task_lock of task.
> + */
> +static int attach_thread_group(struct cgroup *cgrp, char *pidbuf)
> +{
> + struct task_struct *tsk;
> + struct task_struct *t;
> + struct css_set *cg;
> + int ret1, ret2;
> +
> + tsk = attach_get_task(cgrp, pidbuf);
> + if (IS_ERR(tsk))
> +   return PTR_ERR(tsk);
> +
> + /* attach thread group leader */
> + ret1 = cgroup_attach_task(cgrp, tsk, NULL);
> + if (ret1) {
> +   put_task_struct(tsk);
> +   return ret1;
> + }

```

```

> +
> + cg = tsk->cgroups;
> +
> + /* attach all sub-threads */
> + rCU_read_lock();
> + for (t = next_thread(tsk); t != tsk; t = next_thread(t)) {
> +   get_task_struct(t);
> +   ret2 = cgroup_attach_task(cgrp, t, cg);
> +   if (ret2) {
> +     printk(KERN_ERR "cgroup: failed to attach thread %d\n",
> +           (int)task_pid_vnr(t));
> +   ret1 = ret2;
> + }
> +   put_task_struct(t);
> + }
> + rCU_read_unlock();
> +
> + put_task_struct(tsk);
> + return ret1;
> +}
> +
> /* The various types of files and directories in a cgroup file system */
> enum cgroup_filetype {
>   FILE_ROOT,
>   FILE_DIR,
>   FILE_TASKLIST,
> + FILE_PROCLIST,
>   FILE_NOTIFY_ON_RELEASE,
>   FILE_RELEASE_AGENT,
> };
> @@ -1431,6 +1493,9 @@ static ssize_t cgroup_common_file_write(struct cgroup *cgrp,
>   case FILE_TASKLIST:
>     retval = attach_task_by_pid(cgrp, buffer);
>     break;
> + case FILE_PROCLIST:
> +   retval = attach_thread_group(cgrp, buffer);
> +   break;
>   case FILE_NOTIFY_ON_RELEASE:
>     clear_bit(CGRP_RELEASEABLE, &cgrp->flags);
>     if (simple_strtoul(buffer, NULL, 10) != 0)
> @@ -2081,7 +2146,8 @@ struct ctr_struct {
>   * 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 *cgrp)
> +static int pid_array_load(pid_t *pidarray, int npids,
> +  struct cgroup *cgrp, int procs)
> {

```

```

> int n = 0;
> struct cgroup_iter it;
> @@ -2090,6 +2156,8 @@ static int pid_array_load(pid_t *pidarray, int npids, struct cgroup
*cgrp)
>   while ((tsk = cgroup_iter_next(cgrp, &it))) {
>     if (unlikely(n == npids))
>       break;
> +   if (procs && !thread_group_leader(tsk))
> +     continue;
>     pidarray[n++] = task_pid_vnr(tsk);
>   }
>   cgroup_iter_end(cgrp, &it);
> @@ -2178,9 +2246,11 @@ static int pid_array_to_buf(char *buf, int sz, pid_t *a, int npids)
> static int cgroup_tasks_open(struct inode *unused, struct file *file)
> {
>   struct cgroup *cgrp = __d_cgrp(file->f_dentry->d_parent);
> +   struct cftype *cft = __d_cft(file->f_dentry);
>   struct ctr_struct *ctr;
>   pid_t *pidarray;
>   int npids;
> +   int procs;
>   char c;
>
>   if (!(file->f_mode & FMODE_READ))
> @@ -2202,7 +2272,8 @@ static int cgroup_tasks_open(struct inode *unused, struct file *file)
>     if (!pidarray)
>       goto err1;
>
> -   npids = pid_array_load(pidarray, npids, cgrp);
> +   procs = (cft->private == FILE_PROCLIST);
> +   npids = pid_array_load(pidarray, npids, cgrp, procs);
>   sort(pidarray, npids, sizeof(pid_t), cmppid, NULL);
>
>   /* Call pid_array_to_buf() twice, first just to get bufsz */
> @@ -2271,6 +2342,15 @@ static struct cftype files[] = {
> },
>
> {
> +   .name = "procs",
> +   .open = cgroup_tasks_open,
> +   .read = cgroup_tasks_read,
> +   .write = cgroup_common_file_write,
> +   .release = cgroup_tasks_release,
> +   .private = FILE_PROCLIST,
> + },
> +
> + {
> +   .name = "notify_on_release",

```

```
>     .read_u64 = cgroup_read_notify_on_release,
>     .write = cgroup_common_file_write,
> @@ -3002,7 +3082,7 @@ int cgroup_clone(struct task_struct *tsk, struct cgroup_subsys
*> *subsys)
> }
>
> /* All seems fine. Finish by moving the task into the new cgroup */
> - ret = cgroup_attach_task(child, tsk);
> + ret = cgroup_attach_task(child, tsk, NULL);
> mutex_unlock(&cgroup_mutex);
>
> out_release:
> diff --git a/kernel/cpuset.c b/kernel/cpuset.c
> index 039baa4..dec96cf 100644
> --- a/kernel/cpuset.c
> +++ b/kernel/cpuset.c
> @@ -1754,7 +1754,7 @@ static void cpuset_do_move_task(struct task_struct *tsk,
>     struct cpuset_hotplug_scanner *chsp;
>
>     chsp = container_of(scan, struct cpuset_hotplug_scanner, scan);
> - cgroup_attach_task(chsp->to, tsk);
> + cgroup_attach_task(chsp->to, tsk, NULL);
> }
>
> /**
> --
> 1.5.4.rc3
>
>
>
>
```

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