Subject: [RFC][PATCH 1/7] CGroup API: Add cgroup.api control file
Posted by Paul Menage on Fri, 15 Feb 2008 20:44:19 GMT
View Forum Message <> Reply to Message

Add a cgroup.api control file in every cgroup directory. This reports
for each control file the type of data represented by that control
file, and a user-friendly description of the contents.

A secondary effect of this patch is to add the "cgroup." prefix in
front of all cgroup-provided control files. This will reduce the
chance of future control files clashing with user-provided names.

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

---
 include/linux/cgroup.h | 21 +++++++
 kernel/cgroup.c | 133 +++---
 2 files changed, 148 insertions(+), 6 deletions(-)

Index: cgroupmap-2.6.24-mm1/include/linux/cgroup.h
===
--- cgroupmap-2.6.24-mm1.orig/include/linux/cgroup.h
+++ cgroupmap-2.6.24-mm1/include/linux/cgroup.h
@@ -179,12 +179,33 @@ struct css_set {
  * - the 'cftype' of the file is file->f_dentry->d_fsdata
  */

+/*
+ * The various types of control file that are reported in the
+ * cgroup.api file. "String" is a catch-all default, but should only
+ * be used for special cases. If you use the appropriate accessors
+ * (such as "read_uint") in your control file, then you can leave this
+ * as 0 (CGROUP_FILE_UNKNOWN) and let cgroup figure out the right type.
+ */
+enum cgroup_file_type {
+ CGROUP_FILE_UNKNOWN = 0,
+ CGROUP_FILE_VOID,
+ CGROUP_FILE_U64,
+ CGROUP_FILE_STRING,
+};
+
 #define MAX_CFTYPE_NAME 64
 struct cftype {
  /* By convention, the name should begin with the name of the
   * subsystem, followed by a period */
  char name[MAX_CFTYPE_NAME];
  int private;
+

```
+ /* The type of a file - reported in the cgroup.api file */
+ enum cgroup_file_type type;
+
+ /* Human-readable description of the file */
+ const char *desc;
+
  int (*open) (struct inode *inode, struct file *file);
  ssize_t (*read) (struct cgroup *cont, struct cftype *cft,
     struct file *file,
Index: cgroupmap-2.6.24-mm1/kernel/cgroup.c
===================================================================
--- cgroupmap-2.6.24-mm1.orig/kernel/cgroup.c
+++ cgroupmap-2.6.24-mm1/kernel/cgroup.c
@@ -1301,6 +1301,7 @@ enum cgroup_filetype {
  FILE_NOTIFY_ON_RELEASE,
  FILE_RELEASABLE,
  FILE_RELEASE_AGENT,
+ FILE_API,
 };

 static ssize_t cgroup_write_uint(struct cgroup *cgrp, struct cftype *cft,
@@ -1611,17 +1612,21 @@ static int cgroup_create_dir(struct cgro
 }

 int cgroup_add_file(struct cgroup *cgrp,
-         struct cgroup_subsys *subsys,
-         const struct cftype *cft)
+       struct cgroup_subsys *subsys,
+       const struct cftype *cft)
 {
  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, &cgrp->root->flags)) {
-  strcpy(name, subsys->name);
-  strcat(name, ".");
+ if (!test_bit(ROOT_NOPREFIX, &cgrp->root->flags)) {
+  if (subsys) {
+   strcpy(name, subsys->name);
+   strcat(name, ".");
+  } else {
+   strcpy(name, "cgroup.");
+  }
  }
  strcat(name, cft->name);
  BUG_ON(!mutex_is_locked(&dir->d_inode->i_mutex));
```

```
@@ -2126,6 +2131,110 @@ static u64 cgroup_read_releasable(struct
  return test_bit(CGRP_RELEASABLE, &cgrp->flags);
 }

+static const struct file_operations cgroup_api_file_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = seq_release,
+};
+
+/*
+ * cgroup.api is a file in each cgroup directory that gives the types
+ * and descriptions of the various control files in that directory.
+ */
+
+static struct dentry *cgroup_api_advance(struct dentry *d, int advance)
+{
+ struct dentry *parent = d->d_parent;
+ struct list_head *l = &d->d_u.d_child;
+ while (true) {
+  if (advance)
+   l = l->next;
+  advance = true;
+  /* Did we reach the end of the directory? */
+  if (l == &parent->d_subdirs)
+   return NULL;
+  d = container_of(l, struct dentry, d_u.d_child);
+  /* Skip cgroup subdirectories */
+  if (d->d_inode && S_ISREG(d->d_inode->i_mode))
+   return d;
+ }
+}
+
+static void *cgroup_api_start(struct seq_file *sf, loff_t *pos)
+{
+ struct dentry *parent = sf->private;
+ struct dentry *d;
+ loff_t l = 0;
+ spin_lock(&dcache_lock);
+ if (list_empty(&parent->d_subdirs))
+  return NULL;
+ d = container_of(parent->d_subdirs.next, struct dentry, d_u.d_child);
+ d = cgroup_api_advance(d, 0);
+ while (d && l < *pos) {
+  (*pos)++;
+  d = cgroup_api_advance(d, 1);
+ }
+
```

```
+ return d;
+}
+
+static void *cgroup_api_next(struct seq_file *sf, void *v, loff_t *pos)
+{
+ struct dentry *d = v;
+ (*pos)++;
+ return cgroup_api_advance(d, 1);
+}
+
+static void cgroup_api_stop(struct seq_file *sf, void *v)
+{
+ spin_unlock(&dcache_lock);
+}
+
+static const char *cft_type_names[] = {
+ "unknown",
+ "void",
+ "u64",
+ "string",
+ "u64map",
+};
+
+static int cgroup_api_show(struct seq_file *sf, void *v)
+{
+ struct dentry *d = v;
+ struct cftype *cft = __d_cft(d);
+ unsigned type = cft->type;
+ if (type == CGROUP_FILE_UNKNOWN) {
+  if (cft->read_uint)
+   type = CGROUP_FILE_U64;
+  else if (cft->read)
+   type = CGROUP_FILE_STRING;
+  else if (!cft->open)
+   type = CGROUP_FILE_VOID;
+ }
+ if (type >= ARRAY_SIZE(cft_type_names))
+  type = CGROUP_FILE_UNKNOWN;
+ return seq_printf(sf, "%s\t%s\t%s\n", d->d_name.name,
+     cft_type_names[type], cft->desc ?: "");
+}
+
+static const struct seq_operations cgroup_api_seqop = {
+ .start = cgroup_api_start,
+ .next = cgroup_api_next,
+ .stop = cgroup_api_stop,
+ .show = cgroup_api_show,
+};
```

```
+
+static int cgroup_api_open(struct inode *inode, struct file *file)
+{
+ int ret = seq_open(file, &cgroup_api_seqop);
+ if (ret == 0) {
+  struct seq_file *sf = file->private_data;
+  sf->private = file->f_dentry->d_parent;
+  file->f_op = &cgroup_api_file_operations;
+ }
+ return ret;
+}
+
 /*
  * for the common functions, 'private' gives the type of file
  */
@@ -2137,6 +2246,7 @@ static struct cftype files[] = {
   .write = cgroup_common_file_write,
   .release = cgroup_tasks_release,
   .private = FILE_TASKLIST,
+  .desc = "Thread ids of threads in this cgroup",
  },

  {
@@ -2144,13 +2254,23 @@ static struct cftype files[] = {
   .read_uint = cgroup_read_notify_on_release,
   .write = cgroup_common_file_write,
   .private = FILE_NOTIFY_ON_RELEASE,
+  .desc =
+  "Should the release agent trigger when this cgroup is empty",
  },

  {
   .name = "releasable",
   .read_uint = cgroup_read_releasable,
   .private = FILE_RELEASABLE,
- }
+  .desc = "Is this cgroup able to be freed when empty"
+ },
+
+ {
+  .name = "api",
+  .open = cgroup_api_open,
+  .private = FILE_API,
+  .desc = "Control file descriptions",
+ },
 };

 static struct cftype cft_release_agent = {
```

```
@@ -2158,6 +2278,7 @@ static struct cftype cft_release_agent =
  .read = cgroup_common_file_read,
  .write = cgroup_common_file_write,
  .private = FILE_RELEASE_AGENT,
+ .desc = "Path to release agent binary",
 };

 static int cgroup_populate_dir(struct cgroup *cgrp)

--
```