Subject: [PATCH 06/10] CGroup API files: Add cgroup map data type
Posted by Paul Menage on Sat, 23 Feb 2008 22:47:31 GMT
View Forum Message <> Reply to Message

Adds a new type of supported control file representation, a map from
strings to u64 values.

Each map entry is printed as a line in a similar format to
/proc/vmstat, i.e. "$key $value\n"

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

---
 include/linux/cgroup.h | 19 ++++++++++++++++++
 kernel/cgroup.c | 53 ++-
 2 files changed, 71 insertions(+), 1 deletion(-)

Index: cgroup-2.6.25-rc2-mm1/include/linux/cgroup.h
===
--- cgroup-2.6.25-rc2-mm1.orig/include/linux/cgroup.h
+++ cgroup-2.6.25-rc2-mm1/include/linux/cgroup.h
@@ -166,6 +166,16 @@ struct css_set {

 };

+/*
+ * cgroup_map_cb is an abstract callback API for reporting map-valued
+ * control files
+ */
+
+struct cgroup_map_cb {
+ int (*fill)(struct cgroup_map_cb *cb, const char *key, u64 value);
+ void *state;
+};
+
 /* struct cftype:
  *
  * The files in the cgroup filesystem mostly have a very simple read/write
@@ -194,6 +204,15 @@ struct cftype {
   */
  u64 (*read_u64) (struct cgroup *cont, struct cftype *cft);
+ /*
+ * read_map() is used for defining a map of key/value
+ * pairs. It should call cb->fill(cb, key, value) for each
+ * entry. The key/value pairs (and their ordering) should not
+ * change between reboots.
+ */

```
+ int (*read_map) (struct cgroup *cont, struct cftype *cft,
+     struct cgroup_map_cb *cb);
+
  ssize_t (*write) (struct cgroup *cont, struct cftype *cft,
      struct file *file,
      const char __user *buf, size_t nbytes, loff_t *ppos);
Index: cgroup-2.6.25-rc2-mm1/kernel/cgroup.c
===================================================================
--- cgroup-2.6.25-rc2-mm1.orig/kernel/cgroup.c
+++ cgroup-2.6.25-rc2-mm1/kernel/cgroup.c
@@ -1484,6 +1484,46 @@ static ssize_t cgroup_file_read(struct f
  return -EINVAL;
 }

+/*
+ * seqfile ops/methods for returning structured data. Currently just
+ * supports string->u64 maps, but can be extended in future.
+ */
+
+struct cgroup_seqfile_state {
+ struct cftype *cft;
+ struct cgroup *cgroup;
+};
+
+static int cgroup_map_add(struct cgroup_map_cb *cb, const char *key, u64 value)
+{
+ struct seq_file *sf = cb->state;
+ return seq_printf(sf, "%s %llu\n", key, (unsigned long long)value);
+}
+
+static int cgroup_seqfile_show(struct seq_file *m, void *arg)
+{
+ struct cgroup_seqfile_state *state = m->private;
+ struct cftype *cft = state->cft;
+ struct cgroup_map_cb cb = {
+  .fill = cgroup_map_add,
+  .state = m,
+ };
+ return cft->read_map(state->cgroup, cft, &cb);
+}
+
+int cgroup_seqfile_release(struct inode *inode, struct file *file)
+{
+ struct seq_file *seq = file->private_data;
+ kfree(seq->private);
+ return single_release(inode, file);
+}
+
```

```
+static struct file_operations cgroup_seqfile_operations = {
+ .read = seq_read,
+ .llseek = seq_lseek,
+ .release = cgroup_seqfile_release,
+};
+
 static int cgroup_file_open(struct inode *inode, struct file *file)
 {
  int err;
@@ -1496,7 +1536,18 @@ static int cgroup_file_open(struct inode
  cft = __d_cft(file->f_dentry);
  if (!cft)
   return -ENODEV;
- if (cft->open)
+ if (cft->read_map) {
+  struct cgroup_seqfile_state *state =
+   kzalloc(sizeof(*state), GFP_USER);
+  if (!state)
+   return -ENOMEM;
+  state->cft = cft;
+  state->cgroup = __d_cgrp(file->f_dentry->d_parent);
+  file->f_op = &cgroup_seqfile_operations;
+  err = single_open(file, cgroup_seqfile_show, state);
+  if (err < 0)
+   kfree(state);
+ } else if (cft->open)
   err = cft->open(inode, file);
  else
   err = 0;

--

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