Subject: [RFC][PATCH 3/4] container freezer: implement freezer subsystem
Posted by Cedric Le Goater on Wed, 20 Jun 2007 15:08:02 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

This patch implements a new freezer subsystem for the Paul Menage's
container framework. It provides a way to stop and resume execution
of all tasks in a container by writing in the container filesystem.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 include/linux/container_subsys.h | 6
 init/Kconfig | 7
 kernel/Makefile | 1
 kernel/container_freezer.c | 293 ++
 4 files changed, 307 insertions(+)

Index: 2.6.22-rc4-mm2/include/linux/container_subsys.h
===
--- 2.6.22-rc4-mm2.orig/include/linux/container_subsys.h
+++ 2.6.22-rc4-mm2/include/linux/container_subsys.h
@@ -24,3 +24,9 @@ SUBSYS(debug)
 #endif

 /* */
+
+#ifdef CONFIG_CONTAINER_FREEZER
+SUBSYS(freezer)
+#endif
+
+/* */
Index: 2.6.22-rc4-mm2/init/Kconfig
===
--- 2.6.22-rc4-mm2.orig/init/Kconfig
+++ 2.6.22-rc4-mm2/init/Kconfig
@@ -347,6 +347,13 @@ config CONTAINER_CPUACCT
   Provides a simple Resource Controller for monitoring the
   total CPU consumed by the tasks in a container

+config CONTAINER_FREEZER
+   bool "container freezer subsystem"
+   select CONTAINERS
+   help
+   Provides a way to freeze and unfreeze all tasks in a
+   container
+
 config PROC_PID_CPUSET

```
  bool "Include legacy /proc/<pid>/cpuset file"
  depends on CPUSETS
Index: 2.6.22-rc4-mm2/kernel/Makefile
===================================================================
--- 2.6.22-rc4-mm2.orig/kernel/Makefile
+++ 2.6.22-rc4-mm2/kernel/Makefile
@@ -41,6 +41,7 @@ obj-$(CONFIG_CONTAINERS) += container.o
 obj-$(CONFIG_CONTAINER_DEBUG) += container_debug.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CONTAINER_CPUACCT) += cpu_acct.o
+obj-$(CONFIG_CONTAINER_FREEZER) += container_freezer.o
 obj-$(CONFIG_IKCONFIG) += configs.o
 obj-$(CONFIG_STOP_MACHINE) += stop_machine.o
 obj-$(CONFIG_AUDIT) += audit.o auditfilter.o
Index: 2.6.22-rc4-mm2/kernel/container_freezer.c
===================================================================
--- /dev/null
+++ 2.6.22-rc4-mm2/kernel/container_freezer.c
@@ -0,0 +1,293 @@
+/*
+ * container_freezer.c -  container freezer subsystem
+ *
+ * Copyright (C) Cedric Le Goater, IBM Corp. 2007
+ */
+
+#include <linux/module.h>
+#include <linux/container.h>
+#include <linux/fs.h>
+#include <linux/uaccess.h>
+#include <linux/freezer.h>
+
+enum freezer_state {
+ STATE_RUNNING = 0,
+ STATE_FREEZING,
+ STATE_FROZEN,
+};
+
+static const char *freezer_state_strs[] = {
+ "RUNNING\n",
+ "FREEZING\n" ,
+ "FROZEN\n"
+};
+
+struct freezer {
+ struct container_subsys_state css;
+ enum freezer_state state;
+ spinlock_t lock;
+};
```

```
+
+struct container_subsys freezer_subsys;
+
+static inline struct freezer *container_freezer(
+  struct container *container)
+{
+ return container_of(
+  container_subsys_state(container, freezer_subsys_id),
+  struct freezer, css);
+}
+
+static int freezer_create(struct container_subsys *ss,
+      struct container *container)
+{
+ struct freezer *freezer;
+
+ if (!capable(CAP_SYS_ADMIN))
+  return -EPERM;
+
+ freezer = kzalloc(sizeof(struct freezer), GFP_KERNEL);
+ if (!freezer)
+  return -ENOMEM;
+
+ spin_lock_init(&freezer->lock);
+ freezer->state = STATE_RUNNING;
+ container->subsys[freezer_subsys.subsys_id] = &freezer->css;
+ return 0;
+}
+
+static void freezer_destroy(struct container_subsys *ss,
+       struct container *container)
+{
+ struct freezer *freezer;
+
+ freezer = container_freezer(container);
+ kfree(freezer);
+}
+
+
+static int freezer_can_attach(struct container_subsys *ss,
+         struct container *new_container,
+         struct task_struct *task)
+{
+ struct freezer *freezer = container_freezer(new_container);
+ int retval = 0;
+
+ if (freezer->state == STATE_FROZEN)
+  retval = -EBUSY;
```

```
+
+ return retval;
+}
+
+static void freezer_fork(struct container_subsys *ss, struct task_struct *task)
+{
+ struct container *container = task_container(task, freezer_subsys_id);
+ struct freezer *freezer = container_freezer(container);
+
+ spin_lock_irq(&freezer->lock);
+ if (freezer->state == STATE_FREEZING)
+  set_freeze_flag(task);
+ spin_unlock_irq(&freezer->lock);
+}
+
+static int freezer_check_if_frozen(struct container *container)
+{
+ struct container_iter it;
+ struct task_struct *task;
+ unsigned int nfrozen = 0;
+
+ container_iter_start(container, &it);
+
+ while ((task = container_iter_next(container, &it))) {
+  if (frozen(task))
+   nfrozen++;
+ }
+ container_iter_end(container, &it);
+
+ return (nfrozen == container_task_count(container));
+}
+
+static ssize_t freezer_read(struct container *container,
+       struct cftype *cft,
+       struct file *file, char __user *buf,
+       size_t nbytes, loff_t *ppos)
+{
+ struct freezer *freezer = container_freezer(container);
+ enum freezer_state state;
+
+ spin_lock_irq(&freezer->lock);
+ if (freezer->state == STATE_FREEZING)
+  if (freezer_check_if_frozen(container))
+   freezer->state = STATE_FROZEN;
+
+ state = freezer->state;
+ spin_unlock_irq(&freezer->lock);
+
```

```
+ return simple_read_from_buffer(buf, nbytes, ppos,
+           freezer_state_strs[state],
+           strlen(freezer_state_strs[state]) + 1);
+}
+
+static int freezer_kill(struct container *container, int signum)
+{
+ struct container_iter it;
+ struct task_struct *task;
+ int retval = 0;
+
+ container_iter_start(container, &it);
+ while ((task = container_iter_next(container, &it))) {
+  retval = send_sig(signum, task, 1);
+  if (retval)
+   break;
+ }
+
+ container_iter_end(container, &it);
+ return retval;
+}
+
+static int freezer_freeze_tasks(struct container *container)
+{
+ struct container_iter it;
+ struct task_struct *task;
+
+ container_iter_start(container, &it);
+ while ((task = container_iter_next(container, &it)))
+  set_freeze_flag(task);
+
+ container_iter_end(container, &it);
+ return 0;
+}
+
+static int freezer_unfreeze_tasks(struct container *container)
+{
+ struct container_iter it;
+ struct task_struct *task;
+
+ container_iter_start(container, &it);
+ while ((task = container_iter_next(container, &it)))
+  thaw_process(task);
+
+ container_iter_end(container, &it);
+ return 0;
+}
+
```

```
+static int freezer_freeze(struct container *container, int freeze)
+{
+ struct freezer *freezer = container_freezer(container);
+ int retval = 0;
+
+ spin_lock_irq(&freezer->lock);
+ switch (freezer->state) {
+ case STATE_RUNNING:
+  if (freeze) {
+   freezer->state = STATE_FREEZING;
+   retval = freezer_freeze_tasks(container);
+  }
+  break;
+
+ case STATE_FREEZING:
+ case STATE_FROZEN:
+  if (!freeze) {
+   freezer->state = STATE_RUNNING;
+   retval = freezer_unfreeze_tasks(container);
+  }
+  break;
+ }
+ spin_unlock_irq(&freezer->lock);
+
+ return retval;
+}
+
+enum container_filetype {
+ FILE_FREEZE,
+ FILE_KILL,
+};
+
+static ssize_t freezer_write(struct container *container,
+        struct cftype *cft,
+        struct file *file,
+        const char __user *userbuf,
+        size_t nbytes, loff_t *unused_ppos)
+{
+ enum container_filetype type = cft->private;
+ char *buffer;
+ int retval = 0;
+ int value;
+
+ if (nbytes >= PATH_MAX)
+  return -E2BIG;
+
+ /* +1 for nul-terminator */
+ buffer = kmalloc(nbytes + 1, GFP_KERNEL);
```

```
+ if (buffer == NULL)
+  return -ENOMEM;
+
+ if (copy_from_user(buffer, userbuf, nbytes)) {
+  retval = -EFAULT;
+  goto free_buffer;
+ }
+ buffer[nbytes] = 0; /* nul-terminate */
+
+ container_lock();
+
+ if (container_is_removed(container)) {
+  retval = -ENODEV;
+  goto unlock;
+ }
+
+ if (sscanf(buffer, "%d", &value) != 1) {
+  retval = -EIO;
+  goto unlock;
+ }
+
+ switch (type) {
+ case FILE_FREEZE:
+  retval = freezer_freeze(container, value);
+  break;
+
+ case FILE_KILL:
+  retval = freezer_kill(container, value);
+  break;
+ default:
+  retval = -EINVAL;
+ }
+
+ if (retval == 0)
+  retval = nbytes;
+unlock:
+ container_unlock();
+free_buffer:
+ kfree(buffer);
+ return retval;
+}
+
+static struct cftype files[] = {
+ {
+  .name = "freezer.freeze",
+  .read = freezer_read,
+  .write = freezer_write,
+  .private = FILE_FREEZE,
```

```
+ },
+ {
+  .name = "freezer.kill",
+  .write = freezer_write,
+  .private = FILE_KILL,
+ },
+};
+
+static int freezer_populate(struct container_subsys *ss, struct container *cont)
+{
+ return container_add_files(cont, files, ARRAY_SIZE(files));
+}
+
+struct container_subsys freezer_subsys = {
+ .name  = "freezer",
+ .create  = freezer_create,
+ .destroy = freezer_destroy,
+ .populate = freezer_populate,
+ .subsys_id = freezer_subsys_id,
+ .can_attach = freezer_can_attach,
+ .attach  = NULL,
+ .fork  = freezer_fork,
+ .exit  = NULL,
+};

--
```