
Subject: [RFC PATCH 3/4] Container Freezer: Implement freezer cgroup subsystem
Posted by [Matt Helsley](#) on Thu, 03 Apr 2008 21:03:19 GMT
[View Forum Message](#) <> [Reply to Message](#)

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

This is the basic mechanism which should do the right thing for user space tasks in a simple scenario. This will require more work to get the freezing right (cf. `try_to_freeze_tasks()`) for ptraced tasks.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Matt Helsley <matthltc@us.ibm.com>
Tested-by: Matt Helsley <matthltc@us.ibm.com>
Cc: linux-pm@lists.linux-foundation.org

```
include/linux/cgroup_freezer.h | 57 ++++++++
include/linux/cgroup_subsys.h | 6
init/Kconfig                  | 7 +
kernel/Makefile               | 1
kernel/cgroup_freezer.c       | 280 +++++++++++++++++++++++++++++++++++++
kernel/freezer.c              | 1
6 files changed, 352 insertions(+)
```

Index: linux-2.6.25-rc8-mm1/include/linux/cgroup_freezer.h

```
=====
--- /dev/null
+++ linux-2.6.25-rc8-mm1/include/linux/cgroup_freezer.h
@@ -0,0 +1,57 @@
+#ifndef _LINUX_CGROUP_FREEZER_H
+#define _LINUX_CGROUP_FREEZER_H
+/*
+ * cgroup_freezer.h - control group freezer subsystem interface
+ *
+ * Copyright IBM Corp. 2007
+ *
+ * Author : Cedric Le Goater <clg@fr.ibm.com>
+ */
+
+#include <linux/cgroup.h>
+
+#ifdef CONFIG_CGROUP_FREEZER
+
+enum freezer_state {
```

```
+ STATE_RUNNING = 0,  
+ STATE_FREEZING,  
+ STATE_FROZEN,  
+};  
+  
+struct freezer {  
+ struct cgroup_subsys_state css;  
+ enum freezer_state state;  
+ spinlock_t lock;  
+};  
+  
+static inline struct freezer *cgroup_freezer(  
+ struct cgroup *cgroup)  
+{  
+ return container_of(  
+ cgroup_subsys_state(cgroup, freezer_subsys_id),  
+ struct freezer, css);  
+}  
+  
+static inline int cgroup_frozen(struct task_struct *task)  
+{  
+ struct cgroup *cgroup = task_cgroup(task, freezer_subsys_id);  
+ struct freezer *freezer = cgroup_freezer(cgroup);  
+ enum freezer_state state;  
+  
+ spin_lock(&freezer->lock);  
+ state = freezer->state;  
+ spin_unlock(&freezer->lock);  
+  
+ return (state == STATE_FROZEN);  
+}  
+  
+#else /* !CONFIG_CGROUP_FREEZER */  
+  
+static inline int cgroup_frozen(struct task_struct *task)  
+{  
+ return 0;  
+}  
+  
+#endif /* !CONFIG_CGROUP_FREEZER */  
+  
+#endif /* _LINUX_CGROUP_FREEZER_H */
```

Index: linux-2.6.25-rc8-mm1/include/linux/cgroup_subsys.h
=====

-- linux-2.6.25-rc8-mm1.orig/include/linux/cgroup_subsys.h
+++ linux-2.6.25-rc8-mm1/include/linux/cgroup_subsys.h
@@ -46,5 +46,11 @@ SUBSYS(mem_cgroup)
#ifdef CONFIG_CGROUP_DEVICE

=====

```
+++ linux-2.6.25-rc8-mm1/include/linux/cgroup subsys.h
```

SUBSYS(devices)

#endif

/* */

+

+#ifdef CONFIG_CGROUP_FREEZER

+SUBSYS(freezer)

+#endif

+

+/* */

Index: linux-2.6.25-rc8-mm1/init/Kconfig

=====

--- linux-2.6.25-rc8-mm1.orig/init/Kconfig

+++ linux-2.6.25-rc8-mm1/init/Kconfig

@@ -321,10 +321,17 @@ config GROUP_SCHED

default y

help

This feature lets CPU scheduler recognize task groups and control CPU bandwidth allocation to such task groups.

+config CGROUP_FREEZER

+ bool "control group freezer subsystem"

+ depends on CGROUPS

+ help

+ Provides a way to freeze and unfreeze all tasks in a

+ cgroup

+

config FAIR_GROUP_SCHED

bool "Group scheduling for SCHED_OTHER"

depends on GROUP_SCHED

default y

Index: linux-2.6.25-rc8-mm1/kernel/Makefile

=====

--- linux-2.6.25-rc8-mm1.orig/kernel/Makefile

+++ linux-2.6.25-rc8-mm1/kernel/Makefile

@@ -38,10 +38,11 @@ obj-\$(CONFIG_BSD_PROCESS_ACCT) += acct.o

obj-\$(CONFIG_KEXEC) += kexec.o

obj-\$(CONFIG_BACKTRACE_SELF_TEST) += backtracetest.o

obj-\$(CONFIG_COMPAT) += compat.o

obj-\$(CONFIG_CGROUPS) += cgroup.o

obj-\$(CONFIG_CGROUP_DEBUG) += cgroup_debug.o

+obj-\$(CONFIG_CGROUP_FREEZER) += cgroup_freezer.o

obj-\$(CONFIG_CPUSETS) += cpuset.o

obj-\$(CONFIG_CGROUP_NS) += ns_cgroup.o

obj-\$(CONFIG_UTS_NS) += utsname.o

obj-\$(CONFIG_USER_NS) += user_namespace.o

obj-\$(CONFIG_PID_NS) += pid_namespace.o

Index: linux-2.6.25-rc8-mm1/kernel/cgroup_freezer.c

=====

```
--- /dev/null
+++ linux-2.6.25-rc8-mm1/kernel/cgroup_freezer.c
@@ -0,0 +1,280 @@
+/*
+ * cgroup_freezer.c - control group freezer subsystem
+ *
+ * Copyright IBM Corp. 2007
+ *
+ * Author : Cedric Le Goater <clg@fr.ibm.com>
+ */
+
+#include <linux/module.h>
+#include <linux/cgroup.h>
+#include <linux/fs.h>
+#include <linux/uaccess.h>
+#include <linux/freezer.h>
+#include <linux/cgroup_freezer.h>
+
+static const char *freezer_state_strs[] = {
+ "RUNNING\n",
+ "FREEZING\n",
+ "FROZEN\n"
+};
+
+struct cgroup_subsys freezer_subsys;
+
+static struct cgroup_subsys_state *freezer_create(
+ struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ struct freezer *freezer;
+
+ if (!capable(CAP_SYS_ADMIN))
+ return ERR_PTR(-EPERM);
+
+ freezer = kzalloc(sizeof(struct freezer), GFP_KERNEL);
+ if (!freezer)
+ return ERR_PTR(-ENOMEM);
+
+ spin_lock_init(&freezer->lock);
+ freezer->state = STATE_RUNNING;
+ return &freezer->css;
+}
+
+static void freezer_destroy(struct cgroup_subsys *ss,
```

```

+ struct cgroup *cgroup)
+{
+ kfree(cgroup_freezer(cgroup));
+}
+
+
+static int freezer_can_attach(struct cgroup_subsys *ss,
+ struct cgroup *new_cgroup,
+ struct task_struct *task)
+{
+ struct freezer *freezer = cgroup_freezer(new_cgroup);
+ int retval = 0;
+
+ if (freezer->state == STATE_FROZEN)
+ retval = -EBUSY;
+
+ return retval;
+}
+
+static void freezer_fork(struct cgroup_subsys *ss, struct task_struct *task)
+{
+ struct cgroup *cgroup = task_cgroup(task, freezer_subsys_id);
+ struct freezer *freezer = cgroup_freezer(cgroup);
+
+ spin_lock_irq(&freezer->lock);
+ if (freezer->state == STATE_FREEZING)
+ freeze_task(task, 1);
+ spin_unlock_irq(&freezer->lock);
+}
+
+static int freezer_check_if_frozen(struct cgroup *cgroup)
+{
+ struct cgroup_iter it;
+ struct task_struct *task;
+ unsigned int nfrozen = 0;
+
+ cgroup_iter_start(cgroup, &it);
+
+ while ((task = cgroup_iter_next(cgroup, &it))) {
+ if (frozen(task))
+ nfrozen++;
+ }
+ cgroup_iter_end(cgroup, &it);
+
+ return (nfrozen == cgroup_task_count(cgroup));
+}
+
+static ssize_t freezer_read(struct cgroup *cgroup,

```

```

+ struct cftype *cft,
+ struct file *file, char __user *buf,
+ size_t nbytes, loff_t *ppos)
+{
+ struct freezer *freezer = cgroup_freezer(cgroup);
+ enum freezer_state state;
+
+ spin_lock_irq(&freezer->lock);
+ if (freezer->state == STATE_FREEZING)
+ if (freezer_check_if_frozen(cgroup))
+ 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 cgroup *cgroup, int signum)
+{
+ struct cgroup_iter it;
+ struct task_struct *task;
+ int retval = 0;
+
+ cgroup_iter_start(cgroup, &it);
+ while ((task = cgroup_iter_next(cgroup, &it))) {
+     retval = send_sig(signum, task, 1);
+     if (retval)
+         break;
+ }
+
+ cgroup_iter_end(cgroup, &it);
+ return retval;
+}
+
+static int freezer_freeze_tasks(struct cgroup *cgroup)
+{
+ struct cgroup_iter it;
+ struct task_struct *task;
+ unsigned int todo = 0;
+
+ cgroup_iter_start(cgroup, &it);
+ while ((task = cgroup_iter_next(cgroup, &it))) {
+     if (!freeze_task(task, 1))
+         continue;
+ }
+

```

```

+ if (!freezer_should_skip(task))
+   todo++;
+ }
+
+ cgroup_iter_end(cgroup, &it);
+ return todo ? -EBUSY : 0;
+}
+
+static int freezer_unfreeze_tasks(struct cgroup *cgroup)
+{
+   struct cgroup_iter it;
+   struct task_struct *task;
+
+   cgroup_iter_start(cgroup, &it);
+   while ((task = cgroup_iter_next(cgroup, &it)))
+     thaw_process(task);
+
+   cgroup_iter_end(cgroup, &it);
+   return 0;
+}
+
+static int freezer_freeze(struct cgroup *cgroup, int freeze)
+{
+   struct freezer *freezer = cgroup_freezer(cgroup);
+   int retval = 0;
+
+   spin_lock_irq(&freezer->lock);
+   switch (freezer->state) {
+   case STATE_RUNNING:
+     if (freeze) {
+       freezer->state = STATE_FREEZING;
+       retval = freezer_freeze_tasks(cgroup);
+     }
+     break;
+
+   case STATE_FREEZING:
+   case STATE_FROZEN:
+     if (!freeze) {
+       freezer->state = STATE_RUNNING;
+       retval = freezer_unfreeze_tasks(cgroup);
+     }
+     break;
+   }
+   spin_unlock_irq(&freezer->lock);
+
+   return retval;
+}
+

```

```

+enum cgroup_filetype {
+ FILE_FREEZE,
+ FILE_KILL,
+};
+
+static ssize_t freezer_write(struct cgroup *cgroup,
+    struct cftype *cft,
+    struct file *file,
+    const char __user *userbuf,
+    size_t nbytes, loff_t *unused_ppos)
+{
+ enum cgroup_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 */
+
+ cgroup_lock();
+
+ if (cgroup_is_removed(cgroup)) {
+ retval = -ENODEV;
+ goto unlock;
+ }
+
+ if (sscanf(buffer, "%d", &value) != 1) {
+ retval = -EIO;
+ goto unlock;
+ }
+
+ switch (type) {
+ case FILE_FREEZE:
+ retval = freezer_freeze(cgroup, value);
+ break;
+
+ case FILE_KILL:

```

```

+ retval = freezer_kill(cgroup, value);
+ break;
+ default:
+ retval = -EINVAL;
+ }
+
+ if (retval == 0)
+   retval = nbytes;
+unlock:
+ cgroup_unlock();
+free_buffer:
+ kfree(buffer);
+ return retval;
+}
+
+static struct cftype files[] = {
+ {
+   .name = "freeze",
+   .read = freezer_read,
+   .write = freezer_write,
+   .private = FILE_FREEZE,
+ },
+ {
+   .name = "kill",
+   .write = freezer_write,
+   .private = FILE_KILL,
+ },
+};
+
+static int freezer_populate(struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+   return cgroup_add_files(cgroup, ss, files, ARRAY_SIZE(files));
+}
+
+struct cgroup_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,
+};
Index: linux-2.6.25-rc8-mm1/kernel/freezer.c
=====
--- linux-2.6.25-rc8-mm1.orig/kernel/freezer.c

```

```
+++ linux-2.6.25-rc8-mm1/kernel/freezer.c
@@ -120,5 +120,6 @@ int freeze_task(struct task_struct *p, i
    }
    }
    task_unlock(p);
    return ret;
}
+EXPORT_SYMBOL(freeze_task);

--
```

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