Subject: [patch 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Matt Helsley on Tue, 24 Jun 2008 13:58:23 GMT
View Forum Message <> Reply to Message

This patchset reuses the container infrastructure and the swsusp freezer to
freeze a group of tasks.

The freezer subsystem in the container filesystem defines a file named
freezer.state. Writing "FROZEN" to the state file will freeze all tasks in the
cgroup. Subsequently writing "RUNNING" will unfreeze the tasks in the cgroup.
Reading will return the current state.

* Examples of usage :

```
# mkdir /containers/freezer
# mount -t cgroup -ofreezer,signal freezer  /containers
# mkdir /containers/0
# echo $some_pid > /containers/0/tasks
```

to get status of the freezer subsystem :

```
# cat /containers/0/freezer.state
RUNNING
```

to freeze all tasks in the container :

```
# echo FROZEN > /containers/0/freezer.state
# cat /containers/0/freezer.state
FREEZING
# cat /containers/0/freezer.state
FROZEN
```

to unfreeze all tasks in the container :

```
# echo RUNNING > /containers/0/freezer.state
# cat /containers/0/freezer.state
RUNNING
```

to kill all tasks in the container :

```
# echo 9 > /containers/0/signal.kill
```

I've taken Cedric's patches, forward-ported them to 2.6.26-rc5-mm2 + Rafael's
NOSIG patches.

Paul, Pavel asked me to send these to Rafael next. They are patches to make
the freezer useful for checkpoint/restart using cgroups so it would be nice

to get an explicit [N]Ack from you first.

Rafael, if Paul agrees, please consider applying these patches.

Changes since v2:
v3:
 Ported to 2.6.26-rc5-mm2 with Rafael's freezer patches
 Tested on 24 combinations of 3 architectures (x86, x86_64, ppc64)
  with 8 different kernel configs varying power management
  and cgroup config variables. Each patch builds and boots
  in these 24 combinations.
 Passes functional testing.
v2 (roughly patches 3 and 5):
 Moved the "kill" file into a separate cgroup subsystem (signal) and
  it's own patch.
 Changed the name of the file from freezer.freeze to freezer.state.
 Switched from taking 1 and 0 as input to the strings "FROZEN" and
  "RUNNING", respectively. This helps keep the interface
  human-usable if/when we need to more states.
 Checked that stopped or interrupted is "frozen enough"
  Since try_to_freeze() is called upon wakeup of these tasks
  this should be fine. This idea comes from recent changes to
  the freezer.
 Checked that if (task == current) whilst freezing cgroup we're ok
 Fixed bug where -EBUSY would always be returned when freezing
 Added code to handle userspace retries for any remaining -EBUSY

Cheers,
 -Matt Helsley

--

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

---

Subject: [patch 2/4] Container Freezer: Make refrigerator always available
Posted by Matt Helsley on Tue, 24 Jun 2008 13:58:25 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>
Subject: [patch 2/4] Container Freezer: Make refrigerator always available

Now that the TIF_FREEZE flag is available in all architectures,
extract the refrigerator() and freeze_task() from kernel/power/process.c
and make it available to all.

The refrigerator() can now be used in a control group subsystem
implementing a control group freezer.

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>

Changelog:
 Merged Roland's "STOPPED is frozen enough" changes. For details see:
  http://lkml.org/lkml/2008/3/3/676

 include/linux/freezer.h | 24 +++++----
 kernel/Makefile | 2
 kernel/freezer.c | 122 ++
 kernel/power/process.c | 116 --
 4 files changed, 136 insertions(+), 128 deletions(-)

Index: linux-2.6.26-rc5-mm2/include/linux/freezer.h
===
--- linux-2.6.26-rc5-mm2.orig/include/linux/freezer.h
+++ linux-2.6.26-rc5-mm2/include/linux/freezer.h
@@ -4,11 +4,10 @@
 #define FREEZER_H_INCLUDED

 #include <linux/sched.h>
 #include <linux/wait.h>

-#ifdef CONFIG_PM_SLEEP
 /*
  * Check if a process has been frozen
  */
 static inline int frozen(struct task_struct *p)
 {
@@ -37,10 +36,15 @@ static inline void set_freeze_flag(struc
 static inline void clear_freeze_flag(struct task_struct *p)
 {
  clear_tsk_thread_flag(p, TIF_FREEZE);
 }

+static inline bool should_send_signal(struct task_struct *p)
+{
+ return !(p->flags & PF_FREEZER_NOSIG);
+}
+
 /*
  * Wake up a frozen process
  *
  * task_lock() is taken to prevent the race with refrigerator() which may

```
  * occur if the freezing of tasks fails.  Namely, without the lock, if the
@@ -61,22 +65,28 @@ static inline int thaw_process(struct ta
  task_unlock(p);
  return 0;
 }

 extern void refrigerator(void);
-extern int freeze_processes(void);
-extern void thaw_processes(void);

 static inline int try_to_freeze(void)
 {
  if (freezing(current)) {
   refrigerator();
   return 1;
  } else
   return 0;
 }

+extern bool freeze_task(struct task_struct *p, bool sig_only);
+extern void cancel_freezing(struct task_struct *p);
+
+#ifdef CONFIG_PM_SLEEP
+
+extern int freeze_processes(void);
+extern void thaw_processes(void);
+
 /*
  * The PF_FREEZER_SKIP flag should be set by a vfork parent right before it
  * calls wait_for_completion(&vfork) and reset right after it returns from this
  * function.  Next, the parent should call try_to_freeze() to freeze itself
  * appropriately in case the child has exited before the freezing of tasks is
@@ -165,22 +175,14 @@ static inline void set_freezable_with_si
     __retval);     \
  } while (try_to_freeze());     \
  __retval;       \
 })
 #else /* !CONFIG_PM_SLEEP */
-static inline int frozen(struct task_struct *p) { return 0; }
-static inline int freezing(struct task_struct *p) { return 0; }
-static inline void set_freeze_flag(struct task_struct *p) {}
-static inline void clear_freeze_flag(struct task_struct *p) {}
-static inline int thaw_process(struct task_struct *p) { return 1; }

-static inline void refrigerator(void) {}
 static inline int freeze_processes(void) { BUG(); return 0; }
 static inline void thaw_processes(void) {}
```

```
-static inline int try_to_freeze(void) { return 0; }
-
 static inline void freezer_do_not_count(void) {}
 static inline void freezer_count(void) {}
 static inline int freezer_should_skip(struct task_struct *p) { return 0; }
 static inline void set_freezable(void) {}
 static inline void set_freezable_with_signal(void) {}
Index: linux-2.6.26-rc5-mm2/kernel/Makefile
===================================================================
--- linux-2.6.26-rc5-mm2.orig/kernel/Makefile
+++ linux-2.6.26-rc5-mm2/kernel/Makefile
@@ -3,11 +3,11 @@
 #

 obj-y     = sched.o fork.o exec_domain.o panic.o printk.o \
       exit.o itimer.o time.o softirq.o resource.o \
       sysctl.o capability.o ptrace.o timer.o user.o \
-      signal.o sys.o kmod.o workqueue.o pid.o \
+      signal.o sys.o kmod.o workqueue.o pid.o freezer.o \
       rcupdate.o extable.o params.o posix-timers.o \
       kthread.o wait.o kfifo.o sys_ni.o posix-cpu-timers.o mutex.o \
       hrtimer.o rwsem.o nsproxy.o srcu.o semaphore.o \
       notifier.o ksysfs.o pm_qos_params.o sched_clock.o

Index: linux-2.6.26-rc5-mm2/kernel/freezer.c
===================================================================
--- /dev/null
+++ linux-2.6.26-rc5-mm2/kernel/freezer.c
@@ -0,0 +1,122 @@
+/*
+ * kernel/freezer.c - Function to freeze a process
+ *
+ * Originally from kernel/power/process.c
+ */
+
+#include <linux/interrupt.h>
+#include <linux/suspend.h>
+#include <linux/module.h>
+#include <linux/syscalls.h>
+#include <linux/freezer.h>
+
+/*
+ * freezing is complete, mark current process as frozen
+ */
+static inline void frozen_process(void)
+{
+ if (!unlikely(current->flags & PF_NOFREEZE)) {
+  current->flags |= PF_FROZEN;
```

```
+  wmb();
+ }
+ clear_freeze_flag(current);
+}
+
+/* Refrigerator is place where frozen processes are stored :-). */
+void refrigerator(void)
+{
+ /* Hmm, should we be allowed to suspend when there are realtime
+    processes around? */
+ long save;
+
+ task_lock(current);
+ if (freezing(current)) {
+  frozen_process();
+  task_unlock(current);
+ } else {
+  task_unlock(current);
+  return;
+ }
+ save = current->state;
+ pr_debug("%s entered refrigerator\n", current->comm);
+
+ spin_lock_irq(&current->sighand->siglock);
+ recalc_sigpending(); /* We sent fake signal, clean it up */
+ spin_unlock_irq(&current->sighand->siglock);
+
+ for (;;) {
+  set_current_state(TASK_UNINTERRUPTIBLE);
+  if (!frozen(current))
+   break;
+  schedule();
+ }
+ pr_debug("%s left refrigerator\n", current->comm);
+ __set_current_state(save);
+}
+EXPORT_SYMBOL(refrigerator);
+
+static void fake_signal_wake_up(struct task_struct *p)
+{
+ unsigned long flags;
+
+ spin_lock_irqsave(&p->sighand->siglock, flags);
+ signal_wake_up(p, 0);
+ spin_unlock_irqrestore(&p->sighand->siglock, flags);
+}
+
+/**
```

```
+ * freeze_task - send a freeze request to given task
+ * @p: task to send the request to
+ * @sig_only: if set, the request will only be sent if the task has the
+ *  PF_FREEZER_NOSIG flag unset
+ * Return value: 'false', if @sig_only is set and the task has
+ *  PF_FREEZER_NOSIG set or the task is frozen, 'true', otherwise
+ *
+ * The freeze request is sent by setting the tasks's TIF_FREEZE flag and
+ * either sending a fake signal to it or waking it up, depending on whether
+ * or not it has PF_FREEZER_NOSIG set.  If @sig_only is set and the task
+ * has PF_FREEZER_NOSIG set (ie. it is a typical kernel thread), its
+ * TIF_FREEZE flag will not be set.
+ */
+bool freeze_task(struct task_struct *p, bool sig_only)
+{
+ /*
+  * We first check if the task is freezing and next if it has already
+  * been frozen to avoid the race with frozen_process() which first marks
+  * the task as frozen and next clears its TIF_FREEZE.
+  */
+ if (!freezing(p)) {
+  rmb();
+  if (frozen(p))
+   return false;
+
+  if (!sig_only || should_send_signal(p))
+   set_freeze_flag(p);
+  else
+   return false;
+ }
+
+ if (should_send_signal(p)) {
+  if (!signal_pending(p))
+   fake_signal_wake_up(p);
+ } else if (sig_only) {
+  return false;
+ } else {
+  wake_up_state(p, TASK_INTERRUPTIBLE);
+ }
+
+ return true;
+}
+
+void cancel_freezing(struct task_struct *p)
+{
+ unsigned long flags;
+
+ if (freezing(p)) {
```

```
+  pr_debug("  clean up: %s\n", p->comm);
+  clear_freeze_flag(p);
+  spin_lock_irqsave(&p->sighand->siglock, flags);
+  recalc_sigpending_and_wake(p);
+  spin_unlock_irqrestore(&p->sighand->siglock, flags);
+ }
+}
Index: linux-2.6.26-rc5-mm2/kernel/power/process.c
===================================================================
--- linux-2.6.26-rc5-mm2.orig/kernel/power/process.c
+++ linux-2.6.26-rc5-mm2/kernel/power/process.c
@@ -26,125 +26,10 @@ static inline int freezeable(struct task
      (p->exit_state != 0))
   return 0;
  return 1;
 }

-/*
- * freezing is complete, mark current process as frozen
- */
-static inline void frozen_process(void)
-{
- if (!unlikely(current->flags & PF_NOFREEZE)) {
-  current->flags |= PF_FROZEN;
-  wmb();
- }
- clear_freeze_flag(current);
-}
-
-/* Refrigerator is place where frozen processes are stored :-). */
-void refrigerator(void)
-{
- /* Hmm, should we be allowed to suspend when there are realtime
-    processes around? */
- long save;
-
- task_lock(current);
- if (freezing(current)) {
-  frozen_process();
-  task_unlock(current);
- } else {
-  task_unlock(current);
-  return;
- }
- save = current->state;
- pr_debug("%s entered refrigerator\n", current->comm);
-
- spin_lock_irq(&current->sighand->siglock);
```

```
- recalc_sigpending(); /* We sent fake signal, clean it up */
- spin_unlock_irq(&current->sighand->siglock);
-
- for (;;) {
-  set_current_state(TASK_UNINTERRUPTIBLE);
-  if (!frozen(current))
-   break;
-  schedule();
- }
- pr_debug("%s left refrigerator\n", current->comm);
- __set_current_state(save);
-}
-
-static void fake_signal_wake_up(struct task_struct *p)
-{
- unsigned long flags;
-
- spin_lock_irqsave(&p->sighand->siglock, flags);
- signal_wake_up(p, 0);
- spin_unlock_irqrestore(&p->sighand->siglock, flags);
-}
-
-static inline bool should_send_signal(struct task_struct *p)
-{
- return !(p->flags & PF_FREEZER_NOSIG);
-}
-
-/**
- * freeze_task - send a freeze request to given task
- * @p: task to send the request to
- * @sig_only: if set, the request will only be sent if the task has the
- *  PF_FREEZER_NOSIG flag unset
- * Return value: 'false', if @sig_only is set and the task has
- *  PF_FREEZER_NOSIG set or the task is frozen, 'true', otherwise
- *
- * The freeze request is sent by setting the tasks's TIF_FREEZE flag and
- * either sending a fake signal to it or waking it up, depending on whether
- * or not it has PF_FREEZER_NOSIG set.  If @sig_only is set and the task
- * has PF_FREEZER_NOSIG set (ie. it is a typical kernel thread), its
- * TIF_FREEZE flag will not be set.
- */
-static bool freeze_task(struct task_struct *p, bool sig_only)
-{
- /*
-  * We first check if the task is freezing and next if it has already
-  * been frozen to avoid the race with frozen_process() which first marks
-  * the task as frozen and next clears its TIF_FREEZE.
-  */
```

```
- if (!freezing(p)) {
-  rmb();
-  if (frozen(p))
-   return false;
-
-  if (!sig_only || should_send_signal(p))
-   set_freeze_flag(p);
-  else
-   return false;
- }
-
- if (should_send_signal(p)) {
-  if (!signal_pending(p))
-   fake_signal_wake_up(p);
- } else if (sig_only) {
-  return false;
- } else {
-  wake_up_state(p, TASK_INTERRUPTIBLE);
- }
-
- return true;
-}
-
-static void cancel_freezing(struct task_struct *p)
-{
- unsigned long flags;
-
- if (freezing(p)) {
-  pr_debug("  clean up: %s\n", p->comm);
-  clear_freeze_flag(p);
-  spin_lock_irqsave(&p->sighand->siglock, flags);
-  recalc_sigpending_and_wake(p);
-  spin_unlock_irqrestore(&p->sighand->siglock, flags);
- }
-}
-
 static int try_to_freeze_tasks(bool sig_only)
 {
  struct task_struct *g, *p;
  unsigned long end_time;
  unsigned int todo;
@@ -262,6 +147,5 @@ void thaw_processes(void)
  thaw_tasks(false);
  schedule();
  printk("done.\n");
 }

-EXPORT_SYMBOL(refrigerator);
```

--

_____

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

---

Subject: Re: [PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Paul Menage on Tue, 08 Jul 2008 20:06:32 GMT
View Forum Message <> Reply to Message

On Tue, Jul 8, 2008 at 12:39 PM, Matt Helsley <matthltc@us.ibm.com> wrote:
>
> One is to try and disallow users from moving frozen tasks. That doesn't
> seem like a good approach since it would require a new cgroups interface
> "can_detach()".

Detaching from the old cgroup happens at the same time as attaching to
the new cgroup, so can_attach() would work here.

Paul

_____

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

---

Subject: Re: [PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Paul Menage on Tue, 08 Jul 2008 20:07:09 GMT
View Forum Message <> Reply to Message

On Tue, Jul 8, 2008 at 1:06 PM, Paul Menage <menage@google.com> wrote:
> On Tue, Jul 8, 2008 at 12:39 PM, Matt Helsley <matthltc@us.ibm.com> wrote:
>>
>> One is to try and disallow users from moving frozen tasks. That doesn't
>> seem like a good approach since it would require a new cgroups interface
>> "can_detach()".
>
> Detaching from the old cgroup happens at the same time as attaching to
> the new cgroup, so can_attach() would work here.

And the whole can_attach()/attach() protocol needs reworking anyway,
see my email (hopefully) later today.

Paul

_____

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

Subject: Re: [PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Matt Helsley on Wed, 09 Jul 2008 21:58:43 GMT
View Forum Message <> Reply to Message

On Tue, 2008-07-08 at 13:07 -0700, Paul Menage wrote:
> On Tue, Jul 8, 2008 at 1:06 PM, Paul Menage <menage@google.com> wrote:
> > On Tue, Jul 8, 2008 at 12:39 PM, Matt Helsley <matthltc@us.ibm.com> wrote:
> >>
> >> One is to try and disallow users from moving frozen tasks. That doesn't
> >> seem like a good approach since it would require a new cgroups interface
> >> "can_detach()".
> >
> > Detaching from the old cgroup happens at the same time as attaching to
> > the new cgroup, so can_attach() would work here.

Update: I've made a patch implementing this. However it might be better
to just modify attach() to thaw the moving task rather than disallow
moving the frozen task. Serge, Cedric, Kame-san, do you have any
thoughts on which is more useful and/or intuitive?

> And the whole can_attach()/attach() protocol needs reworking anyway,
> see my email (hopefully) later today.
>
> Paul

Interesting. I look forward to seeing this.

Cheers,
 -Matt

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

Subject: Re: [PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by KAMEZAWA Hiroyuki on Thu, 10 Jul 2008 00:39:43 GMT
View Forum Message <> Reply to Message

On Wed, 09 Jul 2008 14:58:43 -0700
Matt Helsley <matthltc@us.ibm.com> wrote:

>
> On Tue, 2008-07-08 at 13:07 -0700, Paul Menage wrote:
> > On Tue, Jul 8, 2008 at 1:06 PM, Paul Menage <menage@google.com> wrote:
> > > On Tue, Jul 8, 2008 at 12:39 PM, Matt Helsley <matthltc@us.ibm.com> wrote:
> > >>
> > >> One is to try and disallow users from moving frozen tasks. That doesn't
> > >> seem like a good approach since it would require a new cgroups interface
> > >> "can_detach()".
> > >
> > > Detaching from the old cgroup happens at the same time as attaching to
> > > the new cgroup, so can_attach() would work here.
>
> Update: I've made a patch implementing this. However it might be better
> to just modify attach() to thaw the moving task rather than disallow
> moving the frozen task. Serge, Cedric, Kame-san, do you have any
> thoughts on which is more useful and/or intuitive?
>

Thank you for explanation in previous mail.

Hmm, just thawing seems atractive but it will confuse people (I think).

I think some kind of process-group is freezed by this freezer and "moving
freezed task" is wrong(unexpected) operation in general. And there will
be no demand to do that from users.
I think just taking "moving freezed task" as error-operation and returning
-EBUSY is better.

Thanks,
-Kame

> > And the whole can_attach()/attach() protocol needs reworking anyway,
> > see my email (hopefully) later today.
> >
> > Paul
>
> Interesting. I look forward to seeing this.
>
> Cheers,
> -Matt
>
>

Subject: [RFC][PATCH] Container Freezer: Don't Let Frozen Stuff Change
Posted by Matt Helsley on Thu, 10 Jul 2008 02:18:29 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-10 at 09:42 +0900, KAMEZAWA Hiroyuki wrote:
> On Wed, 09 Jul 2008 14:58:43 -0700
> Matt Helsley <matthltc@us.ibm.com> wrote:
>
> >
> > On Tue, 2008-07-08 at 13:07 -0700, Paul Menage wrote:
> > > On Tue, Jul 8, 2008 at 1:06 PM, Paul Menage <menage@google.com> wrote:
> > > > On Tue, Jul 8, 2008 at 12:39 PM, Matt Helsley <matthltc@us.ibm.com> wrote:
> > > >>
> > > >> One is to try and disallow users from moving frozen tasks. That doesn't
> > > >> seem like a good approach since it would require a new cgroups interface
> > > >> "can_detach()".
> > > >
> > > > Detaching from the old cgroup happens at the same time as attaching to
> > > > the new cgroup, so can_attach() would work here.
> > >
> > > Update: I've made a patch implementing this. However it might be better
> > > to just modify attach() to thaw the moving task rather than disallow
> > > moving the frozen task. Serge, Cedric, Kame-san, do you have any
> > > thoughts on which is more useful and/or intuitive?
> > >
> >
> > Thank you for explanation in previous mail.
> >
> > Hmm, just thawing seems atractive but it will confuse people (I think).
> >
> > I think some kind of process-group is freezed by this freezer and "moving
> > freezed task" is wrong(unexpected) operation in general. And there will
> > be no demand to do that from users.
> > I think just taking "moving freezed task" as error-operation and returning
> > -EBUSY is better.

Kame-san,

 I've been working on changes to the can_attach() code so it was pretty
easy to try this out.

 Don't let frozen tasks or cgroups change. This means frozen tasks can't
leave their current cgroup for another cgroup. It also means that tasks
cannot be added to or removed from a cgroup in the FROZEN state. We
enforce these rules by checking for frozen tasks and cgroups in the
can_attach() function.

Signed-off-by: Matt Helsley <matthltc@us.ibm.com>

Builds, boots, passes testing against 2.6.26-rc5-mm2

```
 kernel/cgroup_freezer.c |   42 ++++++++++++++++++++++++++----------------
 1 file changed, 25 insertions(+), 17 deletions(-)

Index: linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
===================================================================
--- linux-2.6.26-rc5-mm2.orig/kernel/cgroup_freezer.c
+++ linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
@@ -89,26 +89,43 @@ static void freezer_destroy(struct cgrou
        struct cgroup *cgroup)
 {
  kfree(cgroup_freezer(cgroup));
 }

+/* Task is frozen or will freeze immediately when next it gets woken */
+static bool is_task_frozen_enough(struct task_struct *task)
+{
+ return (frozen(task) || (task_is_stopped_or_traced(task) && freezing(task)));
+}

+/*
+ * The call to cgroup_lock() in the freezer.state write method prevents
+ * a write to that file racing against an attach, and hence the
+ * can_attach() result will remain valid until the attach completes.
+ */
 static int freezer_can_attach(struct cgroup_subsys *ss,
          struct cgroup *new_cgroup,
          struct task_struct *task)
 {
  struct freezer *freezer;
- int retval = 0;
+ int retval;
+
+ /* Anything frozen can't move or be moved to/from */
+
+ if (is_task_frozen_enough(task))
+  return -EBUSY;

- /*
-  * The call to cgroup_lock() in the freezer.state write method prevents
-  * a write to that file racing against an attach, and hence the
-  * can_attach() result will remain valid until the attach completes.
-  */
  freezer = cgroup_freezer(new_cgroup);
  if (freezer->state == STATE_FROZEN)
+  return -EBUSY;
+
```

```
+ retval = 0;
+ task_lock(task);
+ freezer = task_freezer(task);
+ if (freezer->state == STATE_FROZEN)
   retval = -EBUSY;
+ task_unlock(task);
  return retval;
 }

 static void freezer_fork(struct cgroup_subsys *ss, struct task_struct *task)
 {
@@ -139,16 +156,11 @@ static void check_if_frozen(struct cgrou
  unsigned int nfrozen = 0, ntotal = 0;

  cgroup_iter_start(cgroup, &it);
  while ((task = cgroup_iter_next(cgroup, &it))) {
   ntotal++;
-  /*
-   * Task is frozen or will freeze immediately when next it gets
-   * woken
-   */
-  if (frozen(task) ||
-      (task_is_stopped_or_traced(task) && freezing(task)))
+  if (is_task_frozen_enough(task))
    nfrozen++;
  }

  /*
   * Transition to FROZEN when no new tasks can be added ensures
@@ -195,15 +207,11 @@ static int try_to_freeze_cgroup(struct c
  freezer->state = STATE_FREEZING;
  cgroup_iter_start(cgroup, &it);
  while ((task = cgroup_iter_next(cgroup, &it))) {
   if (!freeze_task(task, true))
    continue;
-  if (task_is_stopped_or_traced(task) && freezing(task))
-   /*
-    * The freeze flag is set so these tasks will
-    * immediately go into the fridge upon waking.
-    */
+  if (is_task_frozen_enough(task))
    continue;
   if (!freezing(task) && !freezer_should_skip(task))
    num_cant_freeze_now++;
  }
  cgroup_iter_end(cgroup, &it);
```

Subject: Re: [RFC][PATCH] Container Freezer: Don't Let Frozen Stuff Change
Posted by Li Zefan on Thu, 10 Jul 2008 03:20:12 GMT
View Forum Message <> Reply to Message

Matt Helsley wrote:
> On Thu, 2008-07-10 at 09:42 +0900, KAMEZAWA Hiroyuki wrote:
>> On Wed, 09 Jul 2008 14:58:43 -0700
>> Matt Helsley <matthltc@us.ibm.com> wrote:
>>
>>> On Tue, 2008-07-08 at 13:07 -0700, Paul Menage wrote:
>>>> On Tue, Jul 8, 2008 at 1:06 PM, Paul Menage <menage@google.com> wrote:
>>>>> On Tue, Jul 8, 2008 at 12:39 PM, Matt Helsley <matthltc@us.ibm.com> wrote:
>>>>>> One is to try and disallow users from moving frozen tasks. That doesn't
>>>>>> seem like a good approach since it would require a new cgroups interface
>>>>>> "can_detach()".
>>>>> Detaching from the old cgroup happens at the same time as attaching to
>>>>> the new cgroup, so can_attach() would work here.
>>> Update: I've made a patch implementing this. However it might be better
>>> to just modify attach() to thaw the moving task rather than disallow
>>> moving the frozen task. Serge, Cedric, Kame-san, do you have any
>>> thoughts on which is more useful and/or intuitive?
>>>
>> Thank you for explanation in previous mail.
>>
>> Hmm, just thawing seems atractive but it will confuse people (I think).
>>
>> I think some kind of process-group is freezed by this freezer and "moving
>> freezed task" is wrong(unexpected) operation in general. And there will
>> be no demand to do that from users.
>> I think just taking "moving freezed task" as error-operation and returning
>> -EBUSY is better.
>
> Kame-san,
>
> I've been working on changes to the can_attach() code so it was pretty
> easy to try this out.
>
> Don't let frozen tasks or cgroups change. This means frozen tasks can't
> leave their current cgroup for another cgroup. It also means that tasks
> cannot be added to or removed from a cgroup in the FROZEN state. We
> enforce these rules by checking for frozen tasks and cgroups in the
> can_attach() function.

>
> Signed-off-by: Matt Helsley <matthltc@us.ibm.com>
> ---
> Builds, boots, passes testing against 2.6.26-rc5-mm2
>
> kernel/cgroup_freezer.c | 42 ++++++++++++++++++++++++++----------------
> 1 file changed, 25 insertions(+), 17 deletions(-)
>
> Index: linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
> ===
> --- linux-2.6.26-rc5-mm2.orig/kernel/cgroup_freezer.c
> +++ linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
> @@ -89,26 +89,43 @@ static void freezer_destroy(struct cgrou
>   struct cgroup *cgroup)
> {
>   kfree(cgroup_freezer(cgroup));
> }
>
> +/* Task is frozen or will freeze immediately when next it gets woken */
> +static bool is_task_frozen_enough(struct task_struct *task)
> +{
> + return (frozen(task) || (task_is_stopped_or_traced(task) && freezing(task)));
> +}
>
> +/*
> + * The call to cgroup_lock() in the freezer.state write method prevents
> + * a write to that file racing against an attach, and hence the
> + * can_attach() result will remain valid until the attach completes.
> + */
> static int freezer_can_attach(struct cgroup_subsys *ss,
>   struct cgroup *new_cgroup,
>   struct task_struct *task)
> {
>   struct freezer *freezer;
> - int retval = 0;
> + int retval;
> +
> + /* Anything frozen can't move or be moved to/from */
> +
> + if (is_task_frozen_enough(task))
> + return -EBUSY;
>

cgroup_lock() can prevent the state change of old_cgroup and new_cgroup, but
will the following racy happen ?
  1   2
can_attach(tsk)
 is_task_frozen_enough(tsk) == false

freeze_task(tsk)
attach(tsk)

i.e., will is_task_frozen_enough(tsk) remain valid through can_attach() and attach()?

> - /*
> - * The call to cgroup_lock() in the freezer.state write method prevents
> - * a write to that file racing against an attach, and hence the
> - * can_attach() result will remain valid until the attach completes.
> - */
> freezer = cgroup_freezer(new_cgroup);
> if (freezer->state == STATE_FROZEN)
> + return -EBUSY;
> +
> + retval = 0;
> + task_lock(task);
> + freezer = task_freezer(task);
> + if (freezer->state == STATE_FROZEN)
> retval = -EBUSY;
> + task_unlock(task);
> return retval;
> }
>
> static void freezer_fork(struct cgroup_subsys *ss, struct task_struct *task)
> {
> @@ -139,16 +156,11 @@ static void check_if_frozen(struct cgrou
> unsigned int nfrozen = 0, ntotal = 0;
>
> cgroup_iter_start(cgroup, &it);
> while ((task = cgroup_iter_next(cgroup, &it))) {
> ntotal++;
> - /*
> - * Task is frozen or will freeze immediately when next it gets
> - * woken
> - */
> - if (frozen(task) ||
> - (task_is_stopped_or_traced(task) && freezing(task)))
> + if (is_task_frozen_enough(task))
> nfrozen++;
> }
>
> /*
> * Transition to FROZEN when no new tasks can be added ensures
> @@ -195,15 +207,11 @@ static int try_to_freeze_cgroup(struct c
> freezer->state = STATE_FREEZING;
> cgroup_iter_start(cgroup, &it);
> while ((task = cgroup_iter_next(cgroup, &it))) {
> if (!freeze_task(task, true))

> continue;
> - if (task_is_stopped_or_traced(task) && freezing(task))
> - /*
> - * The freeze flag is set so these tasks will
> - * immediately go into the fridge upon waking.
> - */
> + if (is_task_frozen_enough(task))
> continue;
> if (!freezing(task) && !freezer_should_skip(task))
> num_cant_freeze_now++;
> }
> cgroup_iter_end(cgroup, &it);
>

_____

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

---

Subject: Re: [PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by serue on Thu, 10 Jul 2008 14:40:09 GMT
View Forum Message <> Reply to Message

Quoting KAMEZAWA Hiroyuki (kamezawa.hiroyu@jp.fujitsu.com):
> On Wed, 09 Jul 2008 14:58:43 -0700
> Matt Helsley <matthltc@us.ibm.com> wrote:
>
> >
> > On Tue, 2008-07-08 at 13:07 -0700, Paul Menage wrote:
> > > On Tue, Jul 8, 2008 at 1:06 PM, Paul Menage <menage@google.com> wrote:
> > > > On Tue, Jul 8, 2008 at 12:39 PM, Matt Helsley <matthltc@us.ibm.com> wrote:
> > > >>
> > > >> One is to try and disallow users from moving frozen tasks. That doesn't
> > > >> seem like a good approach since it would require a new cgroups interface
> > > >> "can_detach()".
> > > >
> > > > Detaching from the old cgroup happens at the same time as attaching to
> > > > the new cgroup, so can_attach() would work here.
> > >
> > > Update: I've made a patch implementing this. However it might be better
> > > to just modify attach() to thaw the moving task rather than disallow
> > > moving the frozen task. Serge, Cedric, Kame-san, do you have any
> > > thoughts on which is more useful and/or intuitive?
> > >
> >
> > Thank you for explanation in previous mail.
> >
> > Hmm, just thawing seems atractive but it will confuse people (I think).

>
> I think some kind of process-group is freezed by this freezer and "moving
> freezed task" is wrong(unexpected) operation in general. And there will
> be no demand to do that from users.
> I think just taking "moving freezed task" as error-operation and returning
> -EBUSY is better.
>
> Thanks,
> -Kame

I'm torn. Allowing the moves is kind of cool, but I think I agree that
we should start out with the simpler semantics, which in this case is
disallowing the move. The race Li may have found will only become more
complicated when both sides of the race can change the task's frozen
state.

-serge

Subject: Re: [RFC][PATCH] Container Freezer: Don't Let Frozen Stuff Change
Posted by Matt Helsley on Fri, 11 Jul 2008 23:51:54 GMT
View Forum Message <> Reply to Message

On Thu, 2008-07-10 at 11:20 +0800, Li Zefan wrote:
> Matt Helsley wrote:
> > On Thu, 2008-07-10 at 09:42 +0900, KAMEZAWA Hiroyuki wrote:
> >> On Wed, 09 Jul 2008 14:58:43 -0700
> >> Matt Helsley <matthltc@us.ibm.com> wrote:
> >>
> >>> On Tue, 2008-07-08 at 13:07 -0700, Paul Menage wrote:
> >>>> On Tue, Jul 8, 2008 at 1:06 PM, Paul Menage <menage@google.com> wrote:
> >>>>> On Tue, Jul 8, 2008 at 12:39 PM, Matt Helsley <matthltc@us.ibm.com> wrote:
> >>>>>> One is to try and disallow users from moving frozen tasks. That doesn't
> >>>>>> seem like a good approach since it would require a new cgroups interface
> >>>>>> "can_detach()".
> >>>>> Detaching from the old cgroup happens at the same time as attaching to
> >>>>> the new cgroup, so can_attach() would work here.
> >>> Update: I've made a patch implementing this. However it might be better
> >>> to just modify attach() to thaw the moving task rather than disallow
> >>> moving the frozen task. Serge, Cedric, Kame-san, do you have any
> >>> thoughts on which is more useful and/or intuitive?
> >>>
> >> Thank you for explanation in previous mail.
> >>

> >> Hmm, just thawing seems atractive but it will confuse people (I think).
> >>
> >> I think some kind of process-group is freezed by this freezer and "moving
> >> freezed task" is wrong(unexpected) operation in general. And there will
> >> be no demand to do that from users.
> >> I think just taking "moving freezed task" as error-operation and returning
> >> -EBUSY is better.
> >
> > Kame-san,
> >
> > I've been working on changes to the can_attach() code so it was pretty
> > easy to try this out.
> >
> > Don't let frozen tasks or cgroups change. This means frozen tasks can't
> > leave their current cgroup for another cgroup. It also means that tasks
> > cannot be added to or removed from a cgroup in the FROZEN state. We
> > enforce these rules by checking for frozen tasks and cgroups in the
> > can_attach() function.
> >
> > Signed-off-by: Matt Helsley <matthltc@us.ibm.com>
> > ---
> > Builds, boots, passes testing against 2.6.26-rc5-mm2
> >
> > kernel/cgroup_freezer.c | 42 ++++++++++++++++++++++++++----------------
> > 1 file changed, 25 insertions(+), 17 deletions(-)
> >
> > Index: linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
> > ===
> > --- linux-2.6.26-rc5-mm2.orig/kernel/cgroup_freezer.c
> > +++ linux-2.6.26-rc5-mm2/kernel/cgroup_freezer.c
> > @@ -89,26 +89,43 @@ static void freezer_destroy(struct cgrou
> >   struct cgroup *cgroup)
> > {
> >   kfree(cgroup_freezer(cgroup));
> > }
> >
> > +/* Task is frozen or will freeze immediately when next it gets woken */
> > +static bool is_task_frozen_enough(struct task_struct *task)
> > +{
> > + return (frozen(task) || (task_is_stopped_or_traced(task) && freezing(task)));
> > +}
> >
> > +/*
> > + * The call to cgroup_lock() in the freezer.state write method prevents
> > + * a write to that file racing against an attach, and hence the
> > + * can_attach() result will remain valid until the attach completes.
> > + */
> > static int freezer_can_attach(struct cgroup_subsys *ss,

> >   struct cgroup *new_cgroup,
> >   struct task_struct *task)
> > {
> >   struct freezer *freezer;
> > - int retval = 0;
> > + int retval;
> > +
> > + /* Anything frozen can't move or be moved to/from */
> > +
> > + if (is_task_frozen_enough(task))
> > + return -EBUSY;
> >
>
> cgroup_lock() can prevent the state change of old_cgroup and new_cgroup, but
> will the following racy happen ?
>   1   2

For most of the paths using these functions we have:

cgroup_lock()   cgroup_lock()
...   ...
> can_attach(tsk)
>   is_task_frozen_enough(tsk) == false
>   freeze_task(tsk)
                                     or thaw_process(tsk)
> attach(tsk)
...   ...
cgroup_unlock()   cgroup_unlock()

 I've checked the cgroup freezer subsystem and the cgroup "core" and
this interleaving isn't possible between those two pieces. Only the
swsusp invocation of freeze_task() does not protect freeze/thaw with the
cgroup_lock. I'll be looking into this some more to see if that's really
a problem and if so how we might solve it.

 Thanks for this excellent question.

Cheers,
 -Matt Helsley