
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);
+
+
+
+
+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 ksyzfs.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->sigband->siglock, flags);
+ signal_wake_up(p, 0);
+ spin_unlock_irqrestore(&p->sigband->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 task'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->sigband->siglock, flags);
+ recalc_sigpending_and_wake(p);
+ spin_unlock_irqrestore(&p->sigband->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->sigband->siglock);
- recalc_sigpending(); /* We sent fake signal, clean it up */
- spin_unlock_irq(&current->sigband->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->sigband->siglock, flags);
- signal_wake_up(p, 0);
- spin_unlock_irqrestore(&p->sigband->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 task'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->sigand->siglock, flags);
- recalc_sigpending_and_wake(p);
- spin_unlock_irqrestore(&p->sigand->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>
