Subject: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Matt Helsley on Thu, 03 Apr 2008 21:03:16 GMT
View Forum Message <> Reply to Message

NOTE: Due to problems with my MTA configuration two earlier attempts reached linux-pm but not linux-kernel. Please cc linux-pm@lists.linux-foundation.org on replies.

This patchset is a prototype using the container infrastructure and the swsusp freezer to freeze a group of tasks. I've merely taken Cedric's patches, forward-ported them to 2.6.25-rc8-mm1 and done a small amount of testing.

2 files are defined by the freezer subsystem in the container filesystem :

```
  * "freezer.freeze"

     writing 1 will freeze all tasks and 0 unfreeze
     reading will return the status of the freezer

  * "freezer.kill"

     writing <n> will send signal number <n> to all tasks
```

* Usage :

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

to get status of the freezer subsystem :

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

to freeze all tasks in the container :

```
  # echo 1 > /containers/freezer/0/freezer.freeze
  # cat /containers/freezer/0/freezer.freeze
  FREEZING
  # cat /containers/freezer/0/freezer.freeze
  FROZEN
```

to unfreeze all tasks in the container :

```
  # echo 1 > /containers/freezer/0/freezer.freeze
  # cat /containers/freezer/0/freezer.freeze
  RUNNING
```

to kill all tasks in the container :

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

* Caveats:

  - the FROZEN status is calculated and changed when the container
    file "freezer.freeze" is read.
  - frozen containers will be unfrozen when a system is resumed after
    a suspend. This is addressed by the last patch.

* Series

  Applies to 2.6.25-rc8-mm1

  The first patches make the freezer available to all architectures
  before implementing the freezer subsystem.

[RFC PATCH 1/4] Add TIF_FREEZE flag to all architectures
[RFC PATCH 2/4] Make refrigerator always available
[RFC PATCH 3/4] Implement freezer cgroup subsystem
[RFC PATCH 4/4] Skip frozen cgroups during power management resume

  Each patch compiles, boots, and survives basic LTP containers and controllers
   tests.

Comments are welcome.

Cheers,
 -Matt Helsley

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

Subject: [RFC PATCH 1/4] Container Freezer: Add TIF_FREEZE flag to all architectures
Posted by Matt Helsley on Thu, 03 Apr 2008 21:03:17 GMT

View Forum Message <> Reply to Message

This patch is the first step in making the refrigerator() available
to all architectures, even for those without power management.

The purpose of such a change is to be able to use the refrigerator()
in a new control group subsystem which will implement 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>
Cc: linux-pm@lists.linux-foundation.org

```
---
 include/asm-alpha/thread_info.h     |    2 ++
 include/asm-avr32/thread_info.h     |    2 ++
 include/asm-cris/thread_info.h      |    2 ++
 include/asm-h8300/thread_info.h     |    2 ++
 include/asm-m68k/thread_info.h      |    1 +
 include/asm-m68knommu/thread_info.h |    2 ++
 include/asm-parisc/thread_info.h    |    2 ++
 include/asm-s390/thread_info.h      |    2 ++
 include/asm-sparc/thread_info.h     |    2 ++
 include/asm-sparc64/thread_info.h   |    2 ++
 include/asm-um/thread_info.h        |    2 ++
 include/asm-v850/thread_info.h      |    2 ++
 include/asm-xtensa/thread_info.h    |    2 ++
 13 files changed, 25 insertions(+)

Index: 2.6.25-rc3-mm1/include/asm-alpha/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-alpha/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-alpha/thread_info.h
@@ -76,12 +76,14 @@ register struct thread_info *__current_t
 #define TIF_UAC_SIGBUS  7
 #define TIF_MEMDIE  8
 #define TIF_RESTORE_SIGMASK 9 /* restore signal mask in do_signal */
+#define TIF_FREEZE  19 /* is freezing for suspend */

 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
 #define _TIF_SIGPENDING  (1<<TIF_SIGPENDING)
 #define _TIF_NEED_RESCHED (1<<TIF_NEED_RESCHED)
 #define _TIF_POLLING_NRFLAG (1<<TIF_POLLING_NRFLAG)
 #define _TIF_RESTORE_SIGMASK (1<<TIF_RESTORE_SIGMASK)
+#define _TIF_FREEZE  (1<<TIF_FREEZE)

 /* Work to do on interrupt/exception return.  */
 #define _TIF_WORK_MASK  (_TIF_SIGPENDING | _TIF_NEED_RESCHED)
Index: 2.6.25-rc3-mm1/include/asm-avr32/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-avr32/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-avr32/thread_info.h
@@ -88,6 +88,7 @@ static inline struct thread_info *curren
```

```
 #define TIF_MEMDIE  6
 #define TIF_RESTORE_SIGMASK 7 /* restore signal mask in do_signal */
 #define TIF_CPU_GOING_TO_SLEEP 8 /* CPU is entering sleep 0 mode */
+#define TIF_FREEZE  19 /* is freezing for suspend */
 #define TIF_DEBUG  30 /* debugging enabled */
 #define TIF_USERSPACE  31      /* true if FS sets userspace */

@@ -99,6 +100,7 @@ static inline struct thread_info *curren
 #define _TIF_MEMDIE  (1 << TIF_MEMDIE)
 #define _TIF_RESTORE_SIGMASK (1 << TIF_RESTORE_SIGMASK)
 #define _TIF_CPU_GOING_TO_SLEEP (1 << TIF_CPU_GOING_TO_SLEEP)
+#define _TIF_FREEZE  (1 << TIF_FREEZE)

 /* Note: The masks below must never span more than 16 bits! */

Index: 2.6.25-rc3-mm1/include/asm-cris/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-cris/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-cris/thread_info.h
@@ -86,6 +86,7 @@ struct thread_info {
 #define TIF_RESTORE_SIGMASK 9 /* restore signal mask in do_signal() */
 #define TIF_POLLING_NRFLAG 16 /* true if poll_idle() is polling TIF_NEED_RESCHED */
 #define TIF_MEMDIE  17
+#define TIF_FREEZE  19 /* is freezing for suspend */

 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
 #define _TIF_NOTIFY_RESUME (1<<TIF_NOTIFY_RESUME)
@@ -93,6 +94,7 @@ struct thread_info {
 #define _TIF_NEED_RESCHED (1<<TIF_NEED_RESCHED)
 #define _TIF_RESTORE_SIGMASK (1<<TIF_RESTORE_SIGMASK)
 #define _TIF_POLLING_NRFLAG (1<<TIF_POLLING_NRFLAG)
+#define _TIF_FREEZE  (1<<TIF_FREEZE)

 #define _TIF_WORK_MASK  0x0000FFFE /* work to do on interrupt/exception return */
 #define _TIF_ALLWORK_MASK 0x0000FFFF /* work to do on any return to u-space */
Index: 2.6.25-rc3-mm1/include/asm-h8300/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-h8300/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-h8300/thread_info.h
@@ -92,6 +92,7 @@ static inline struct thread_info *curren
         TIF_NEED_RESCHED */
 #define TIF_MEMDIE  4
 #define TIF_RESTORE_SIGMASK 5 /* restore signal mask in do_signal() */
+#define TIF_FREEZE  19 /* is freezing for suspend */

 /* as above, but as bit values */
 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
@@ -99,6 +100,7 @@ static inline struct thread_info *curren
```

```
 #define _TIF_NEED_RESCHED (1<<TIF_NEED_RESCHED)
 #define _TIF_POLLING_NRFLAG (1<<TIF_POLLING_NRFLAG)
 #define _TIF_RESTORE_SIGMASK (1<<TIF_RESTORE_SIGMASK)
+#define _TIF_FREEZE  (1<<TIF_FREEZE)

 #define _TIF_WORK_MASK  0x0000FFFE /* work to do on interrupt/exception return */

Index: 2.6.25-rc3-mm1/include/asm-m68k/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-m68k/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-m68k/thread_info.h
@@ -58,5 +58,6 @@ struct thread_info {
 #define TIF_DELAYED_TRACE 14 /* single step a syscall */
 #define TIF_SYSCALL_TRACE 15 /* syscall trace active */
 #define TIF_MEMDIE  16
+#define TIF_FREEZE  19

 #endif /* _ASM_M68K_THREAD_INFO_H */
Index: 2.6.25-rc3-mm1/include/asm-m68knommu/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-m68knommu/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-m68knommu/thread_info.h
@@ -88,12 +88,14 @@ static inline struct thread_info *curren
 #define TIF_POLLING_NRFLAG 3 /* true if poll_idle() is polling
         TIF_NEED_RESCHED */
 #define TIF_MEMDIE  4
+#define TIF_FREEZE  19 /* is freezing for suspend */

 /* as above, but as bit values */
 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
 #define _TIF_SIGPENDING  (1<<TIF_SIGPENDING)
 #define _TIF_NEED_RESCHED (1<<TIF_NEED_RESCHED)
 #define _TIF_POLLING_NRFLAG (1<<TIF_POLLING_NRFLAG)
+#define _TIF_FREEZE  (1<<TIF_FREEZE)

 #define _TIF_WORK_MASK  0x0000FFFE /* work to do on interrupt/exception return */

Index: 2.6.25-rc3-mm1/include/asm-parisc/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-parisc/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-parisc/thread_info.h
@@ -62,6 +62,7 @@ struct thread_info {
 #define TIF_32BIT               4       /* 32 bit binary */
 #define TIF_MEMDIE  5
 #define TIF_RESTORE_SIGMASK 6 /* restore saved signal mask */
+#define TIF_FREEZE  19 /* is freezing for suspend */

 #define _TIF_SYSCALL_TRACE (1 << TIF_SYSCALL_TRACE)
 #define _TIF_SIGPENDING  (1 << TIF_SIGPENDING)
@@ -69,6 +70,7 @@ struct thread_info {
 #define _TIF_POLLING_NRFLAG (1 << TIF_POLLING_NRFLAG)
 #define _TIF_32BIT  (1 << TIF_32BIT)
 #define _TIF_RESTORE_SIGMASK (1 << TIF_RESTORE_SIGMASK)
+#define _TIF_FREEZE  (1 << TIF_FREEZE)

 #define _TIF_USER_WORK_MASK     (_TIF_SIGPENDING | \
                                  _TIF_NEED_RESCHED | _TIF_RESTORE_SIGMASK)
Index: 2.6.25-rc3-mm1/include/asm-s390/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-s390/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-s390/thread_info.h
@@ -101,6 +101,7 @@ static inline struct thread_info *curren
        TIF_NEED_RESCHED */
 #define TIF_31BIT  18 /* 32bit process */
 #define TIF_MEMDIE  19
+#define TIF_FREEZE  20 /* is freezing for suspend */

 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
 #define _TIF_RESTORE_SIGMASK (1<<TIF_RESTORE_SIGMASK)
@@ -113,6 +114,7 @@ static inline struct thread_info *curren
 #define _TIF_USEDFPU  (1<<TIF_USEDFPU)
 #define _TIF_POLLING_NRFLAG (1<<TIF_POLLING_NRFLAG)
 #define _TIF_31BIT  (1<<TIF_31BIT)
+#define _TIF_FREEZE  (1<<TIF_FREEZE)

 #endif /* __KERNEL__ */

Index: 2.6.25-rc3-mm1/include/asm-sparc/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-sparc/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-sparc/thread_info.h
@@ -137,6 +137,7 @@ BTFIXUPDEF_CALL(void, free_thread_info,
 #define TIF_POLLING_NRFLAG 9 /* true if poll_idle() is polling
      * TIF_NEED_RESCHED */
 #define TIF_MEMDIE  10
+#define TIF_FREEZE  19 /* is freezing for suspend */

 /* as above, but as bit values */
 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
@@ -145,6 +146,7 @@ BTFIXUPDEF_CALL(void, free_thread_info,
 #define _TIF_RESTORE_SIGMASK (1<<TIF_RESTORE_SIGMASK)
 #define _TIF_USEDFPU  (1<<TIF_USEDFPU)
 #define _TIF_POLLING_NRFLAG (1<<TIF_POLLING_NRFLAG)
+#define _TIF_FREEZE  (1<<TIF_FREEZE)

 #endif /* __KERNEL__ */
```

```
Index: 2.6.25-rc3-mm1/include/asm-sparc64/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-sparc64/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-sparc64/thread_info.h
@@ -236,6 +236,7 @@ register struct thread_info *current_thr
 #define TIF_ABI_PENDING  12
 #define TIF_MEMDIE  13
 #define TIF_POLLING_NRFLAG 14
+#define TIF_FREEZE  19 /* is freezing for suspend */

 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
 #define _TIF_SIGPENDING  (1<<TIF_SIGPENDING)
@@ -249,6 +250,7 @@ register struct thread_info *current_thr
 #define _TIF_RESTORE_SIGMASK (1<<TIF_RESTORE_SIGMASK)
 #define _TIF_ABI_PENDING (1<<TIF_ABI_PENDING)
 #define _TIF_POLLING_NRFLAG (1<<TIF_POLLING_NRFLAG)
+#define _TIF_FREEZE  (1<<TIF_FREEZE)

 #define _TIF_USER_WORK_MASK ((0xff << TI_FLAG_WSAVED_SHIFT) | \
      (_TIF_SIGPENDING | _TIF_RESTORE_SIGMASK | \
Index: 2.6.25-rc3-mm1/include/asm-um/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-um/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-um/thread_info.h
@@ -83,6 +83,7 @@ static inline struct thread_info *curren
 #define TIF_MEMDIE   5
 #define TIF_SYSCALL_AUDIT 6
 #define TIF_RESTORE_SIGMASK 7
+#define TIF_FREEZE  19 /* is freezing for suspend */

 #define _TIF_SYSCALL_TRACE (1 << TIF_SYSCALL_TRACE)
 #define _TIF_SIGPENDING  (1 << TIF_SIGPENDING)
@@ -91,5 +92,6 @@ static inline struct thread_info *curren
 #define _TIF_MEMDIE  (1 << TIF_MEMDIE)
 #define _TIF_SYSCALL_AUDIT (1 << TIF_SYSCALL_AUDIT)
 #define _TIF_RESTORE_SIGMASK (1 << TIF_RESTORE_SIGMASK)
+#define _TIF_FREEZE  (1 << TIF_FREEZE)

 #endif
Index: 2.6.25-rc3-mm1/include/asm-v850/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-v850/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-v850/thread_info.h
@@ -82,12 +82,14 @@ struct thread_info {
 #define TIF_POLLING_NRFLAG 3 /* true if poll_idle() is polling
        TIF_NEED_RESCHED */
 #define TIF_MEMDIE  4
```

```
+#define TIF_FREEZE  19 /* is freezing for suspend */

 /* as above, but as bit values */
 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
 #define _TIF_SIGPENDING  (1<<TIF_SIGPENDING)
 #define _TIF_NEED_RESCHED (1<<TIF_NEED_RESCHED)
 #define _TIF_POLLING_NRFLAG (1<<TIF_POLLING_NRFLAG)
+#define _TIF_FREEZE  (1<<TIF_FREEZE)


 /* Size of kernel stack for each process.  */
Index: 2.6.25-rc3-mm1/include/asm-xtensa/thread_info.h
===================================================================
--- 2.6.25-rc3-mm1.orig/include/asm-xtensa/thread_info.h
+++ 2.6.25-rc3-mm1/include/asm-xtensa/thread_info.h
@@ -138,6 +138,7 @@ static inline struct thread_info *curren
 #define TIF_MEMDIE  5
 #define TIF_RESTORE_SIGMASK 6 /* restore signal mask in do_signal() */
 #define TIF_POLLING_NRFLAG 16 /* true if poll_idle() is polling TIF_NEED_RESCHED */
+#define TIF_FREEZE  19 /* is freezing for suspend */

 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
 #define _TIF_SIGPENDING  (1<<TIF_SIGPENDING)
@@ -146,6 +147,7 @@ static inline struct thread_info *curren
 #define _TIF_IRET  (1<<TIF_IRET)
 #define _TIF_POLLING_NRFLAG (1<<TIF_POLLING_NRFLAG)
 #define _TIF_RESTORE_SIGMASK (1<<TIF_RESTORE_SIGMASK)
+#define _TIF_FREEZE  (1<<TIF_FREEZE)

 #define _TIF_WORK_MASK  0x0000FFFE /* work to do on interrupt/exception return */
 #define _TIF_ALLWORK_MASK 0x0000FFFF /* work to do on any return to u-space */

--

--
```

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

Subject: [RFC PATCH 2/4] Container Freezer: Make refrigerator always available
Posted by Matt Helsley on Thu, 03 Apr 2008 21:03:18 GMT

View Forum Message <> Reply to Message

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>
Cc: linux-pm@lists.linux-foundation.org

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

```
 include/linux/freezer.h |   19 ++-----
 kernel/Makefile         |    2
 kernel/freezer.c        |  124 ++++++++++++++++++++++++++++++++++++++++++++++++
 kernel/power/process.c  |  113 -------------------------------------------
 4 files changed, 132 insertions(+), 126 deletions(-)
```

```
Index: linux-2.6.25-rc8-mm1/include/linux/freezer.h
===================================================================
--- linux-2.6.25-rc8-mm1.orig/include/linux/freezer.h
+++ linux-2.6.25-rc8-mm1/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)
 {
@@ -61,22 +60,27 @@ 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 int freeze_task(struct task_struct *p, int with_mm_only);
+
+#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
@@ -156,22 +160,13 @@ static inline void set_freezable(void)
     __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) {}

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
@@ -7,11 +7,11 @@ obj-y     = sched.o fork.o exec_domain.o
      sysctl.o capability.o ptrace.o timer.o user.o \
      signal.o sys.o kmod.o workqueue.o pid.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
+     notifier.o ksysfs.o pm_qos_params.o freezer.o

 obj-$(CONFIG_SYSCTL_SYSCALL_CHECK) += sysctl_check.o
 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
 obj-$(CONFIG_DEBUG_MUTEXES) += mutex-debug.o
Index: linux-2.6.25-rc8-mm1/kernel/freezer.c
===================================================================
--- /dev/null
+++ linux-2.6.25-rc8-mm1/kernel/freezer.c
@@ -0,0 +1,124 @@
+/*
+ * 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);
+}
+
+static int has_mm(struct task_struct *p)
+{
+ return (p->mm && !(p->flags & PF_BORROWED_MM));
+}
+
+/**
+ * freeze_task - send a freeze request to given task
+ * @p: task to send the request to
+ * @with_mm_only: if set, the request will only be sent if the task has its
+ *  own mm
+ * Return value: 0, if @with_mm_only is set and the task has no mm of its
+ *  own or the task is frozen, 1, otherwise
+ *
+ * The freeze request is sent by seting 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 its own mm (ie. it is a user land task).  If @with_mm_only
+ * is set and the task has no mm of its own (ie. it is a kernel thread),
+ * its TIF_FREEZE flag should not be set.
+ *
+ * The task_lock() is necessary to prevent races with exit_mm() or
+ * use_mm()/unuse_mm() from occuring.
+ */
+int freeze_task(struct task_struct *p, int with_mm_only)
+{
+ int ret = 1;
+
+ task_lock(p);
+ if (freezing(p)) {
+  if (has_mm(p)) {
+   if (!signal_pending(p))
+    fake_signal_wake_up(p);
+  } else {
+   if (with_mm_only)
+    ret = 0;
+   else
+    wake_up_state(p, TASK_INTERRUPTIBLE);
+  }
+ } else {
+  rmb();
+  if (frozen(p)) {
+   ret = 0;
+  } else {
+   if (has_mm(p)) {
+    set_freeze_flag(p);
+    fake_signal_wake_up(p);
+   } else {
+    if (with_mm_only) {
+     ret = 0;
+    } else {
+     set_freeze_flag(p);
+     wake_up_state(p, TASK_INTERRUPTIBLE);
+    }
+   }
+  }
+ }
+ task_unlock(p);
+ return ret;
+}
Index: linux-2.6.25-rc8-mm1/kernel/power/process.c
===================================================================
--- linux-2.6.25-rc8-mm1.orig/kernel/power/process.c
+++ linux-2.6.25-rc8-mm1/kernel/power/process.c
@@ -29,121 +29,10 @@ 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 int has_mm(struct task_struct *p)
-{
- return (p->mm && !(p->flags & PF_BORROWED_MM));
-}
-
-/**
- * freeze_task - send a freeze request to given task
- * @p: task to send the request to
- * @with_mm_only: if set, the request will only be sent if the task has its
- *  own mm
- * Return value: 0, if @with_mm_only is set and the task has no mm of its
- *  own or the task is frozen, 1, otherwise
- *
- * The freeze request is sent by seting 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 its own mm (ie. it is a user land task).  If @with_mm_only
- * is set and the task has no mm of its own (ie. it is a kernel thread),
- * its TIF_FREEZE flag should not be set.
- *
- * The task_lock() is necessary to prevent races with exit_mm() or
- * use_mm()/unuse_mm() from occuring.
- */
-static int freeze_task(struct task_struct *p, int with_mm_only)
-{
- int ret = 1;
-
- task_lock(p);
- if (freezing(p)) {
-  if (has_mm(p)) {
-   if (!signal_pending(p))
-    fake_signal_wake_up(p);
-  } else {
-   if (with_mm_only)
-    ret = 0;
-   else
-    wake_up_state(p, TASK_INTERRUPTIBLE);
-  }
- } else {
-  rmb();
-  if (frozen(p)) {
-   ret = 0;
-  } else {
-   if (has_mm(p)) {
-    set_freeze_flag(p);
-    fake_signal_wake_up(p);
-   } else {
-    if (with_mm_only) {
-     ret = 0;
-    } else {
-     set_freeze_flag(p);
-     wake_up_state(p, TASK_INTERRUPTIBLE);
-    }
-   }
-  }
- }
- task_unlock(p);
- return ret;
-}

 static void cancel_freezing(struct task_struct *p)
 {
  unsigned long flags;

@@ -274,7 +163,5 @@ void thaw_processes(void)
  thaw_tasks(FREEZER_KERNEL_THREADS);
  thaw_tasks(FREEZER_USER_SPACE);
  schedule();
  printk("done.\n");
 }
-
-EXPORT_SYMBOL(refrigerator);
```

--

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
 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

---

Subject: Re: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Paul Menage on Thu, 03 Apr 2008 23:49:24 GMT
View Forum Message <> Reply to Message

On Thu, Apr 3, 2008 at 2:03 PM, <matthltc@us.ibm.com> wrote:

```
>
>    * "freezer.kill"
>
>      writing <n> will send signal number <n> to all tasks
>
```

My first thought (not having looked at the code yet) is that sending a signal doesn't really have anything to do with freezing, so it shouldn't be in the same subsystem. Maybe a separate subsystem called "signal"?

And more than that, it's not something that requires any particular per-process state, so there's no reason that the subsystem that provides the "kill" functionality shouldn't be able to be mounted in multiple hierarchies.

How about if I added support for stateless subsystems, that could potentially be mounted in multiple hierarchies at once? They wouldn't need an entry in the css set, since they have no state.

```
>  * Usage :
>
>    # mkdir /containers/freezer
>    # mount -t container -ofreezer freezer  /containers/freezer
>    # mkdir /containers/freezer/0
>    # echo $some_pid > /containers/freezer/0/tasks
>
>  to get status of the freezer subsystem :
>
>    # cat /containers/freezer/0/freezer.freeze
>    RUNNING
>
>  to freeze all tasks in the container :
>
>    # echo 1 > /containers/freezer/0/freezer.freeze
```

>   # cat /containers/freezer/0/freezer.freeze
>   FREEZING
>   # cat /containers/freezer/0/freezer.freeze
>   FROZEN

Could we separate this out into two files? One called "freeze" that's
a 0/1 for whether we're intending to freeze the subsystem, and one
called "frozen" that indicates whether it is frozen? And maybe a
"state" file to report the RUNNING/FREEZING/FROZEN distinction in a
human-readable way?

Paul

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

Subject: Re: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Matt Helsley on Fri, 04 Apr 2008 03:03:00 GMT
View Forum Message <> Reply to Message

On Thu, 2008-04-03 at 16:49 -0700, Paul Menage wrote:
> On Thu, Apr 3, 2008 at 2:03 PM, <matthltc@us.ibm.com> wrote:
> >
> >   * "freezer.kill"
> >
> >   writing <n> will send signal number <n> to all tasks
> >
>
> My first thought (not having looked at the code yet) is that sending a
> signal doesn't really have anything to do with freezing, so it
> shouldn't be in the same subsystem. Maybe a separate subsystem called
> "signal"?
>
> And more than that, it's not something that requires any particular
> per-process state, so there's no reason that the subsystem that
> provides the "kill" functionality shouldn't be able to be mounted in
> multiple hierarchies.
>
> How about if I added support for stateless subsystems, that could
> potentially be mounted in multiple hierarchies at once? They wouldn't
> need an entry in the css set, since they have no state.

This seems reasonable to me. A quick look at Cedric's patches suggests
there's no need for such cgroup subsystems to be tied together -- the
signalling is all done internally to the freeze_task(), refrigerator(),
and thaw_process() functions from what I recall.

```
> >  * Usage :
> >
> >    # mkdir /containers/freezer
> >    # mount -t container -ofreezer freezer  /containers/freezer
> >    # mkdir /containers/freezer/0
> >    # echo $some_pid > /containers/freezer/0/tasks
> >
> >  to get status of the freezer subsystem :
> >
> >    # cat /containers/freezer/0/freezer.freeze
> >    RUNNING
> >
> >  to freeze all tasks in the container :
> >
> >    # echo 1 > /containers/freezer/0/freezer.freeze
> >    # cat /containers/freezer/0/freezer.freeze
> >    FREEZING
> >    # cat /containers/freezer/0/freezer.freeze
> >    FROZEN
>
```

> Could we separate this out into two files? One called "freeze" that's
> a 0/1 for whether we're intending to freeze the subsystem, and one
> called "frozen" that indicates whether it is frozen? And maybe a
> "state" file to report the RUNNING/FREEZING/FROZEN distinction in a
> human-readable way?

3 files seems like overkill. I think making them human-readable is good and can be done with two files: "state" (read-only) and "state-next" (read/write). Transitions between RUNNING and FROZEN are obvious when state-next != state. I think the advantages are it's pretty human-readable, you don't need separate strings and files for the transitions, it's clear what's about to happen (IMHO), and it only requires 2 files. Some examples:

To initiate freezing:

```
# cat /containers/freezer/0/freezer.state
RUNNING
# echo "FROZEN" > /containers/freezer/0/freezer.state-next
# cat /containers/freezer/0/freezer.state
RUNNING
# cat /containers/freezer/0/freezer.state-next
FROZEN
# sleep N
# cat /containers/freezer/0/freezer.state
FROZEN
# cat /containers/freezer/0/freezer.state-next
```

FROZEN

So to cancel freezing you might see something like:

```
# cat /containers/freezer/0/freezer.state
RUNNING
# cat /containers/freezer/0/freezer.state-next
FROZEN
# echo "RUNNING" > /containers/freezer/0/freezer.state-next
# cat /containers/freezer/0/freezer.state-next
RUNNING
```

If you wanted to know if a group was transitioning:

```
# diff /containers/freezer/0/freezer.state /containers/freezer/0/freezer.state-next
```

Or:

```
# current=`cat /containers/freezer/0/freezer.state`
# next=`cat /containers/freezer/0/freezer.state-next`
# [ "$current" != "$next" ] && echo "Transitioning"
# [ "$current" == "RUNNING" -a "$next" == "FROZEN" ] && echo "Freezing"
# [ "$current" == "FROZEN" -a "$next" == "RUNNING" ] && echo "Thawing"
# [ "$current" == "RUNNING" -a "$next" == "RUNNING" ] && echo "No-op"
# [ "$current" == "FROZEN" -a "$next" == "FROZEN" ] && echo "No-op"
```

etc.

Cheers,
 -Matt Helsley

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

Subject: Re: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by serue on Fri, 04 Apr 2008 14:11:06 GMT

View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):
> On Thu, Apr 3, 2008 at 2:03 PM, <matthltc@us.ibm.com> wrote:
> >
> >   * "freezer.kill"
> >
> >   writing <n> will send signal number <n> to all tasks
> >
>

> My first thought (not having looked at the code yet) is that sending a
> signal doesn't really have anything to do with freezing, so it
> shouldn't be in the same subsystem. Maybe a separate subsystem called
> "signal"?
>
> And more than that, it's not something that requires any particular
> per-process state, so there's no reason that the subsystem that
> provides the "kill" functionality shouldn't be able to mounted in
> multiple hierarchies.
>
> How about if I added support for stateless subsystems, that could
> potentially be mounted in multiple hierarchies at once? They wouldn't
> need an entry in the css set, since they have no state.
>
> > * Usage :
> >
> >   # mkdir /containers/freezer
> >   # mount -t container -ofreezer freezer /containers/freezer
> >   # mkdir /containers/freezer/0
> >   # echo $some_pid > /containers/freezer/0/tasks
> >
> > to get status of the freezer subsystem :
> >
> >   # cat /containers/freezer/0/freezer.freeze
> >   RUNNING
> >
> > to freeze all tasks in the container :
> >
> >   # echo 1 > /containers/freezer/0/freezer.freeze
> >   # cat /containers/freezer/0/freezer.freeze
> >   FREEZING
> >   # cat /containers/freezer/0/freezer.freeze
> >   FROZEN
>
> Could we separate this out into two files? One called "freeze" that's
> a 0/1 for whether we're intending to freeze the subsystem, and one
> called "frozen" that indicates whether it is frozen? And maybe a
> "state" file to report the RUNNING/FREEZING/FROZEN distinction in a
> human-readable way?

One thing Oren had mentioned for checkpoint/restart was having more
states - i.e. restoring, checkpointing... So then (assuming we used
this subsys for that) we'd have more than the two files. Which is
probably fine, just wanted to point that out.

-serge

____
Containers mailing list

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

Subject: Re: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Oren Laadan on Fri, 04 Apr 2008 15:56:43 GMT
View Forum Message <> Reply to Message

```
Matt Helsley wrote:
> On Thu, 2008-04-03 at 16:49 -0700, Paul Menage wrote:
>> On Thu, Apr 3, 2008 at 2:03 PM,  <matthltc@us.ibm.com> wrote:
>>>    * "freezer.kill"
>>>
>>>      writing <n> will send signal number <n> to all tasks
>>>
>> My first thought (not having looked at the code yet) is that sending a
>> signal doesn't really have anything to do with freezing, so it
>> shouldn't be in the same subsystem. Maybe a separate subsystem called
>> "signal"?
>>
>> And more than that, it's not something that requires any particular
>> per-process state, so there's no reason that the subsystem that
>> provides the "kill" functionality shouldn't be able to be mounted in
>> multiple hierarchies.
>>
>> How about if I added support for stateless subsystems, that could
>> potentially be mounted in multiple hierarchies at once? They wouldn't
>> need an entry in the css set, since they have no state.
>
>  This seems reasonable to me. A quick look at Cedric's patches suggests
> there's no need for such cgroup subsystems to be tied together -- the
> signalling is all done internally to the freeze_task(), refrigerator(),
> and thaw_process() functions from what I recall.
>
>>>  * Usage :
>>>
>>>    # mkdir /containers/freezer
>>>    # mount -t container -ofreezer freezer  /containers/freezer
>>>    # mkdir /containers/freezer/0
>>>    # echo $some_pid > /containers/freezer/0/tasks
>>>
>>>  to get status of the freezer subsystem :
>>>
>>>    # cat /containers/freezer/0/freezer.freeze
>>>    RUNNING
>>>
>>>  to freeze all tasks in the container :
>>>
```

```
>>>    # echo 1 > /containers/freezer/0/freezer.freeze
>>>    # cat /containers/freezer/0/freezer.freeze
>>>    FREEZING
>>>    # cat /containers/freezer/0/freezer.freeze
>>>    FROZEN
>> Could we separate this out into two files? One called "freeze" that's
>> a 0/1 for whether we're intending to freeze the subsystem, and one
>> called "frozen" that indicates whether it is frozen? And maybe a
>> "state" file to report the RUNNING/FREEZING/FROZEN distinction in a
>> human-readable way?
>
> 3 files seems like overkill. I think making them human-readable is good
> and can be done with two files: "state" (read-only) and
> "state-next" (read/write). Transitions between RUNNING and FROZEN are
> obvious when state-next != state. I think the advantages are it's pretty
> human-readable, you don't need separate strings and files for the
> transitions, it's clear what's about to happen (IMHO), and it only
> requires 2 files. Some examples:
>
> To initiate freezing:
>
> # cat /containers/freezer/0/freezer.state
> RUNNING
> # echo "FROZEN" > /containers/freezer/0/freezer.state-next
> # cat /containers/freezer/0/freezer.state
> RUNNING
> # cat /containers/freezer/0/freezer.state-next
> FROZEN
> # sleep N
> # cat /containers/freezer/0/freezer.state
> FROZEN
> # cat /containers/freezer/0/freezer.state-next
> FROZEN
>
> So to cancel freezing you might see something like:
>
> # cat /containers/freezer/0/freezer.state
> RUNNING
> # cat /containers/freezer/0/freezer.state-next
> FROZEN
> # echo "RUNNING" > /containers/freezer/0/freezer.state-next
> # cat /containers/freezer/0/freezer.state-next
> RUNNING
>
> If you wanted to know if a group was transitioning:
>
> # diff /containers/freezer/0/freezer.state /containers/freezer/0/freezer.state-next
>
```

```
> Or:
> # current=`cat /containers/freezer/0/freezer.state`
> # next=`cat /containers/freezer/0/freezer.state-next`
> # [ "$current" != "$next" ] && echo "Transitioning"
> # [ "$current" == "RUNNING" -a "$next" == "FROZEN" ] && echo "Freezing"
> # [ "$current" == "FROZEN" -a "$next" == "RUNNING" ] && echo "Thawing"
> # [ "$current" == "RUNNING" -a "$next" == "RUNNING" ] && echo "No-op"
> # [ "$current" == "FROZEN" -a "$next" == "FROZEN" ] && echo "No-op"
```

First, I totally agree with Serge's comment (oh well, it's about my own suggestion, so I must) - for checkpoint/restart we'll need more states if we are to use the same subsystem.

Second, my gut feeling is that a single, atomic operation to get the status is preferred over multiple (non-atomic) operations. In other words, I suggest a single state file instead of two. You can encode every possible transition in a single state. It's not that the kernel doesn't know what's going on inside, so it can just as well report it directly. I don't see the benefit of using two files.

Oren.

>
> etc.
>
> Cheers,
> -Matt Helsley
>
> ____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers

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

Subject: Re: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Matt Helsley on Fri, 04 Apr 2008 22:27:40 GMT

View Forum Message <> Reply to Message

On Fri, 2008-04-04 at 11:56 -0400, Oren Laadan wrote:
>
> Matt Helsley wrote:
> > On Thu, 2008-04-03 at 16:49 -0700, Paul Menage wrote:
> >> On Thu, Apr 3, 2008 at 2:03 PM, <matthltc@us.ibm.com> wrote:
> >>>   * "freezer.kill"

```
> >>>
> >>>     writing <n> will send signal number <n> to all tasks
> >>>
> >> My first thought (not having looked at the code yet) is that sending a
> >> signal doesn't really have anything to do with freezing, so it
> >> shouldn't be in the same subsystem. Maybe a separate subsystem called
> >> "signal"?
> >>
> >> And more than that, it's not something that requires any particular
> >> per-process state, so there's no reason that the subsystem that
> >> provides the "kill" functionality shouldn't be able to be mounted in
> >> multiple hierarchies.
> >>
> >> How about if I added support for stateless subsystems, that could
> >> potentially be mounted in multiple hierarchies at once? They wouldn't
> >> need an entry in the css set, since they have no state.
> >
> >  This seems reasonable to me. A quick look at Cedric's patches suggests
> > there's no need for such cgroup subsystems to be tied together -- the
> > signalling is all done internally to the freeze_task(), refrigerator(),
> > and thaw_process() functions from what I recall.
> >
> >>>  * Usage :
> >>>
> >>>    # mkdir /containers/freezer
> >>>    # mount -t container -ofreezer freezer  /containers/freezer
> >>>    # mkdir /containers/freezer/0
> >>>    # echo $some_pid > /containers/freezer/0/tasks
> >>>
> >>>  to get status of the freezer subsystem :
> >>>
> >>>    # cat /containers/freezer/0/freezer.freeze
> >>>    RUNNING
> >>>
> >>>  to freeze all tasks in the container :
> >>>
> >>>    # echo 1 > /containers/freezer/0/freezer.freeze
> >>>    # cat /containers/freezer/0/freezer.freeze
> >>>    FREEZING
> >>>    # cat /containers/freezer/0/freezer.freeze
> >>>    FROZEN
> >> Could we separate this out into two files? One called "freeze" that's
> >> a 0/1 for whether we're intending to freeze the subsystem, and one
> >> called "frozen" that indicates whether it is frozen? And maybe a
> >> "state" file to report the RUNNING/FREEZING/FROZEN distinction in a
> >> human-readable way?
> >
> > 3 files seems like overkill. I think making them human-readable is good
```

> > and can be done with two files: "state" (read-only) and
> > "state-next" (read/write). Transitions between RUNNING and FROZEN are
> > obvious when state-next != state. I think the advantages are it's pretty
> > human-readable, you don't need separate strings and files for the
> > transitions, it's clear what's about to happen (IMHO), and it only
> > requires 2 files. Some examples:
> >
> > To initiate freezing:
> >

```
> > # cat /containers/freezer/0/freezer.state
> > RUNNING
> > # echo "FROZEN" > /containers/freezer/0/freezer.state-next
> > # cat /containers/freezer/0/freezer.state
> > RUNNING
> > # cat /containers/freezer/0/freezer.state-next
> > FROZEN
> > # sleep N
> > # cat /containers/freezer/0/freezer.state
> > FROZEN
> > # cat /containers/freezer/0/freezer.state-next
> > FROZEN
```

> >
> > So to cancel freezing you might see something like:
> >

```
> > # cat /containers/freezer/0/freezer.state
> > RUNNING
> > # cat /containers/freezer/0/freezer.state-next
> > FROZEN
> > # echo "RUNNING" > /containers/freezer/0/freezer.state-next
> > # cat /containers/freezer/0/freezer.state-next
> > RUNNING
```

> >
> > If you wanted to know if a group was transitioning:
> >

```
> > # diff /containers/freezer/0/freezer.state /containers/freezer/0/freezer.state-next
```

> >
> > Or:

```
> > # current=`cat /containers/freezer/0/freezer.state`
> > # next=`cat /containers/freezer/0/freezer.state-next`
> > # [ "$current" != "$next" ] && echo "Transitioning"
> > # [ "$current" == "RUNNING" -a "$next" == "FROZEN" ] && echo "Freezing"
> > # [ "$current" == "FROZEN" -a "$next" == "RUNNING" ] && echo "Thawing"
> > # [ "$current" == "RUNNING" -a "$next" == "RUNNING" ] && echo "No-op"
> > # [ "$current" == "FROZEN" -a "$next" == "FROZEN" ] && echo "No-op"
```

>
> First, I totally agree with Serge's comment (oh well, it's about my
> own suggestion, so I must) - for checkpoint/restart we'll need more
> states if we are to use the same subsystem.

I don't have an upper limit on how many more states we will need and I think that number impacts the interface significantly. Can you give us an estimate?

> Second, my gut feeling is that a single, atomic operation to get the
> status is preferred over multiple (non-atomic) operations. In other
> words, I suggest a single state file instead of two. You can encode
> every possible transition in a single state. It's not that the kernel

If the transitions are to be human-readable and there are more than a small number of states it may not be desirable to encode transitions as states. Paul's reason for suggesting the additional file(s), as best I could tell, was to keep the interface human-readable.

Cheers,
-Matt Helsley

____

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

Subject: Re: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Oren Laadan on Sat, 05 Apr 2008 00:30:33 GMT

View Forum Message <> Reply to Message

Matt Helsley wrote:
> On Fri, 2008-04-04 at 11:56 -0400, Oren Laadan wrote:
>> Matt Helsley wrote:
>>> On Thu, 2008-04-03 at 16:49 -0700, Paul Menage wrote:
>>>> On Thu, Apr 3, 2008 at 2:03 PM, <matthltc@us.ibm.com> wrote:
>>>>>   * "freezer.kill"
>>>>>
>>>>>   writing <n> will send signal number <n> to all tasks
>>>>>
>>>> My first thought (not having looked at the code yet) is that sending a
>>>> signal doesn't really have anything to do with freezing, so it
>>>> shouldn't be in the same subsystem. Maybe a separate subsystem called
>>>> "signal"?
>>>>
>>>> And more than that, it's not something that requires any particular
>>>> per-process state, so there's no reason that the subsystem that
>>>> provides the "kill" functionality shouldn't be able to be mounted in
>>>> multiple hierarchies.
>>>>

>>>> How about if I added support for stateless subsystems, that could
>>>> potentially be mounted in multiple hierarchies at once? They wouldn't
>>>> need an entry in the css set, since they have no state.
>>> This seems reasonable to me. A quick look at Cedric's patches suggests
>>> there's no need for such cgroup subsystems to be tied together -- the
>>> signalling is all done internally to the freeze_task(), refrigerator(),
>>> and thaw_process() functions from what I recall.
>>>
>>>>> * Usage :
>>>>>
>>>>>   # mkdir /containers/freezer
>>>>>   # mount -t container -ofreezer freezer /containers/freezer
>>>>>   # mkdir /containers/freezer/0
>>>>>   # echo $some_pid > /containers/freezer/0/tasks
>>>>>
>>>>> to get status of the freezer subsystem :
>>>>>
>>>>>   # cat /containers/freezer/0/freezer.freeze
>>>>>   RUNNING
>>>>>
>>>>> to freeze all tasks in the container :
>>>>>
>>>>>   # echo 1 > /containers/freezer/0/freezer.freeze
>>>>>   # cat /containers/freezer/0/freezer.freeze
>>>>>   FREEZING
>>>>>   # cat /containers/freezer/0/freezer.freeze
>>>>>   FROZEN
>>>> Could we separate this out into two files? One called "freeze" that's
>>>> a 0/1 for whether we're intending to freeze the subsystem, and one
>>>> called "frozen" that indicates whether it is frozen? And maybe a
>>>> "state" file to report the RUNNING/FREEZING/FROZEN distinction in a
>>>> human-readable way?
>>> 3 files seems like overkill. I think making them human-readable is good
>>> and can be done with two files: "state" (read-only) and
>>> "state-next" (read/write). Transitions between RUNNING and FROZEN are
>>> obvious when state-next != state. I think the advantages are it's pretty
>>> human-readable, you don't need separate strings and files for the
>>> transitions, it's clear what's about to happen (IMHO), and it only
>>> requires 2 files. Some examples:
>>>
>>> To initiate freezing:
>>>
>>> # cat /containers/freezer/0/freezer.state
>>> RUNNING
>>> # echo "FROZEN" > /containers/freezer/0/freezer.state-next
>>> # cat /containers/freezer/0/freezer.state
>>> RUNNING
>>> # cat /containers/freezer/0/freezer.state-next

```
>>> FROZEN
>>> # sleep N
>>> # cat /containers/freezer/0/freezer.state
>>> FROZEN
>>> # cat /containers/freezer/0/freezer.state-next
>>> FROZEN
>>>
>>> So to cancel freezing you might see something like:
>>>
>>> # cat /containers/freezer/0/freezer.state
>>> RUNNING
>>> # cat /containers/freezer/0/freezer.state-next
>>> FROZEN
>>> # echo "RUNNING" > /containers/freezer/0/freezer.state-next
>>> # cat /containers/freezer/0/freezer.state-next
>>> RUNNING
>>>
>>> If you wanted to know if a group was transitioning:
>>>
>>> # diff /containers/freezer/0/freezer.state /containers/freezer/0/freezer.state-next
>>>
>>> Or:
>>> # current=`cat /containers/freezer/0/freezer.state`
>>> # next=`cat /containers/freezer/0/freezer.state-next`
>>> # [ "$current" != "$next" ] && echo "Transitioning"
>>> # [ "$current" == "RUNNING" -a "$next" == "FROZEN" ] && echo "Freezing"
>>> # [ "$current" == "FROZEN" -a "$next" == "RUNNING" ] && echo "Thawing"
>>> # [ "$current" == "RUNNING" -a "$next" == "RUNNING" ] && echo "No-op"
>>> # [ "$current" == "FROZEN" -a "$next" == "FROZEN" ] && echo "No-op"
>> First, I totally agree with Serge's comment (oh well, it's about my
>> own suggestion, so I must) - for checkpoint/restart we'll need more
>> states if we are to use the same subsystem.
>
>  I don't have an upper limit on how many more states we will need and I
> think that number impacts the interface significantly. Can you give us
> an estimate?
```

In Zap there are (using the current terminology):
RUNNING - running
FREEZING - transition to FROZEN
FROZEN - frozen
THAWING - transition to RUNNING
CKPTING - being checkpointed (needs special semantics* for some ops)
RSTRTING - being restarted (may need special semantics* for some ops)
ABORTING - transition to DEAD (mainly when aborting a failed restart)
DEAD - dead (but not fully cleaned up yet)

There is also "SPECIAL" in which some operations are not allowed;

this simplifies dealing with a bunch of races related to checkpoint/
restart, but I'm not sure it's a must. If anything, it only stays
for very short times (like an uninterruptible sleep) saying "don't
mess with this container now, it's busy".

I have very good justifications for almost all the states, a good
reasoning for DEAD, and a case for SPECIAL (although there may be
a way to do without it).

Despite the "many" states, there are very few transitions: CKPTING
can only be reached from- and changed to- FROZEN. A similar rule holds
for RSTRTING. ABORTING is reached from RSTRTING, and leads to DEAD.
The only one I don't cover is reaching DEAD from any other state
(except SPECIAL) but I never saw a reason to explicitly encode that.
Something like this (without SPECIAL):

```
        ->  FREEZING  ->        <-> CKPTING
RUNNING                FROZEN
        <-  THAWING   <-        <-> RSTRTING -> ABORTING -> DEAD
```

Out of curiosity - why does the number of states impact the interface
so much ?

>
>> Second, my gut feeling is that a single, atomic operation to get the
>> status is preferred over multiple (non-atomic) operations. In other
>> words, I suggest a single state file instead of two. You can encode
>> every possible transition in a single state. It's not that the kernel
>
> If the transitions are to be human-readable and there are more than a
> small number of states it may not be desirable to encode transitions as
> states. Paul's reason for suggesting the additional file(s), as best I
> could tell, was to keep the interface human-readable.

The scheme above has very few transitions. Whatever be the final scheme,
I would prefer not to have many possible transition (the full matrix).
It probably isn't necessary either.

The main idea behind limiting the transitions above, is that checkpoint
requires to first freeze the container to be able to capture a consistent
view of the state of its processes; that means, for example, that we also
would like to prevent signals from being delivered to tasks in a frozen
state (if you do want to signal - thaw it first).

There is also the issue of a pre-checkpoint (a.k.a live migration) where
significant state (mainly memory) of the container is recorded while the
container is still running, and when it's finally frozen little state
remains to be saved, reducing the application downtime. I didn't see a

need for a special state for this case; instead Zap uses a status flag
that belongs to the container.

Oren.

>
> Cheers,
> -Matt Helsley
>
>

Subject: Re: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Matt Helsley on Sat, 05 Apr 2008 00:54:03 GMT

View Forum Message <> Reply to Message

On Fri, 2008-04-04 at 20:30 -0400, Oren Laadan wrote:
>
> Matt Helsley wrote:

<snip>

> > I don't have an upper limit on how many more states we will need and I
> > think that number impacts the interface significantly. Can you give us
> > an estimate?
>
> In Zap there are (using the current terminology):
> RUNNING - running
> FREEZING - transition to FROZEN
> FROZEN - frozen
> THAWING - transition to RUNNING
> CKPTING - being checkpointed (needs special semantics* for some ops)
> RSTRTING - being restarted (may need special semantics* for some ops)
> ABORTING - transition to DEAD (mainly when aborting a failed restart)
> DEAD - dead (but not fully cleaned up yet)
>
> There is also "SPECIAL" in which some operations are not allowed;
> this simplifies dealing with a bunch of races related to checkpoint/
> restart, but I'm not sure it's a must. If anything, it only stays
> for very short times (like an uninterruptible sleep) saying "don't
> mess with this container now, it's busy".
>
> I have very good justifications for almost all the states, a good

> reasoning for DEAD, and a case for SPECIAL (although there may be
> a way to do without it).
>
> Despite the "many" states, there are very few transitions: CKPTING
> can only be reached from- and changed to- FROZEN. A similar rule holds
> for RSTRTING. ABORTING is reached from RSTRTING, and leads to DEAD.
> The only one I don't cover is reaching DEAD from any other state
> (except SPECIAL) but I never saw a reason to explicitly encode that.
> Something like this (without SPECIAL):
>

```
>         ->  FREEZING  ->        <-> CKPTING
> RUNNING                FROZEN
>         <-  THAWING   <-        <-> RSTRTING -> ABORTING -> DEAD
```

>
> Out of curiosity - why does the number of states impact the interface
> so much ?

Only that, if you're trying to keep the interface human readable, then
having too many names for things (states and transitions) can make the
interface less than intuitive. Doesn't look like that's an issue for the
above state machine though so I think you're right -- a single file with
transitions encoded as states seems best.

> >
> >> Second, my gut feeling is that a single, atomic operation to get the
> >> status is preferred over multiple (non-atomic) operations. In other
> >> words, I suggest a single state file instead of two. You can encode
> >> every possible transition in a single state. It's not that the kernel
> >
> > If the transitions are to be human-readable and there are more than a
> > small number of states it may not be desirable to encode transitions as
> > states. Paul's reason for suggesting the additional file(s), as best I
> > could tell, was to keep the interface human-readable.
>
> The scheme above has very few transitions. Whatever be the final scheme,
> I would prefer not to have many possible transition (the full matrix).
> It probably isn't necessary either.
>
> The main idea behind limiting the transitions above, is that checkpoint
> requires to first freeze the container to be able to capture a consistent
> view of the state of its processes; that means, for example, that we also
> would like to prevent signals from being delivered to tasks in a frozen
> state (if you do want to signal - thaw it first).
>
> There is also the issue of a pre-checkpoint (a.k.a live migration) where
> significant state (mainly memory) of the container is recorded while the
> container is still running, and when it's finally frozen little state
> remains to be saved, reducing the application downtime. I didn't see a

> need for a special state for this case; instead Zap uses a status flag
> that belongs to the container.
>
> Oren.

Thanks for the explanation of the states.

Cheers,
 -Matt Helsley

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

Subject: Re: [RFC PATCH 0/4] Container Freezer: Reuse Suspend Freezer
Posted by Pavel Machek on Fri, 11 Apr 2008 11:49:22 GMT

View Forum Message <> Reply to Message

Hi!

> NOTE: Due to problems with my MTA configuration two earlier attempts reached linux-pm
> but not linux-kernel. Please cc linux-pm@lists.linux-foundation.org on replies.
>
> This patchset is a prototype using the container infrastructure and
> the swsusp freezer to freeze a group of tasks. I've merely taken Cedric's
> patches, forward-ported them to 2.6.25-rc8-mm1 and done a small amount of
> testing.

Okay, freezer probably does what you want, but be warned that Linus is
not exactly in love with freezer. You probably can get away with using
it for user processes, but maybe you should drop him the line saying
you want to expand freezer usage and see what happens.
        Pavel
--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html
____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: Re: [RFC PATCH 1/4] Container Freezer: Add TIF_FREEZE flag to all architectures
Posted by Pavel Machek on Fri, 11 Apr 2008 11:49:30 GMT

View Forum Message <> Reply to Message

On Thu 2008-04-03 14:03:17, matthltc@us.ibm.com wrote:
> This patch is the first step in making the refrigerator() available
> to all architectures, even for those without power management.
>
> The purpose of such a change is to be able to use the refrigerator()
> in a new control group subsystem which will implement 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>
> Cc: linux-pm@lists.linux-foundation.org

ACK.
Pavel
--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

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

Subject: Re: [RFC PATCH 3/4] Container Freezer: Implement freezer cgroup subsystem
Posted by Pavel Machek on Fri, 11 Apr 2008 11:49:47 GMT

View Forum Message <> Reply to Message

Hi!

> 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.

```
> --- /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>
> + */
```

If you have copyright, add GPL.

```
> --- /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>
> + */
```

Same here.

```
>+static struct cgroup_subsys_state *freezer_create(
>+	struct cgroup_subsys *ss, struct cgroup *cgroup)
>+{
```

Function headers are somehow non-traditional.

```
+	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;
+}
```

One space too many after "return" :-).

Hmm, returning pointer inside struct freezer is rather ugly, right?
Could you just pass struct freezer around?
	Pavel

--
(english) http://www.livejournal.com/~pavelmachek
(cesky, pictures) http://atrey.karlin.mff.cuni.cz/~pavel/picture/horses/blog.html

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