Subject: [RFC][PATCH 0/5] Container Freezer: Reuse Suspend Freezer
Posted by Matt Helsley on Thu, 24 Apr 2008 06:47:56 GMT
View Forum Message <> Reply to Message

This patchset reuses 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-mm1 and tested the expected common cases.

Changes since v1:
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

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/freezer
   # mkdir /containers/freezer/0
   # echo $some_pid > /containers/freezer/0/tasks

to get status of the freezer subsystem :

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

to freeze all tasks in the container :

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

to unfreeze all tasks in the container :

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

to kill all tasks in the container :

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

* Caveats:

- The cgroup moves into the FROZEN state once all tasks in the cgroup are frozen. This is calculated and changed when the container file "freezer.state" is read or written.
- Frozen containers will be unfrozen when a system is resumed after a suspend. This is addressed by a subsequent patch.

* Series

Applies to 2.6.25-mm1

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

[RFC PATCH 1/5] Add TIF_FREEZE flag to all architectures
[RFC PATCH 2/5] Make refrigerator always available
[RFC PATCH 3/5] Implement freezer cgroup subsystem
[RFC PATCH 4/5] Skip frozen cgroups during power management resume
[RFC PATCH 5/5] Implement signal cgroup subsytem

Comments are welcome. I'm planning to finish up testing with ptrace'd and vforking processes and then, if it still seems appropriate, resubmit as a non-RFC series next.

Cheers,
-Matt Helsley

--

Subject: [RFC][PATCH 1/5] Container Freezer: Add TIF_FREEZE flag to all

architectures
Posted by Matt Helsley on Thu, 24 Apr 2008 06:47:57 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>
---
 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: linux-2.6.25-mm1/include/asm-alpha/thread_info.h
===
--- linux-2.6.25-mm1.orig/include/asm-alpha/thread_info.h
+++ linux-2.6.25-mm1/include/asm-alpha/thread_info.h
@@ -74,16 +74,18 @@ register struct thread_info *__current_t
 #define TIF_UAC_NOPRINT 5 /* see sysinfo.h */
 #define TIF_UAC_NOFIX 6
 #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)

 /* Work to do on any return to userspace.  */
Index: linux-2.6.25-mm1/include/asm-avr32/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-avr32/thread_info.h
+++ linux-2.6.25-mm1/include/asm-avr32/thread_info.h
@@ -86,10 +86,11 @@ static inline struct thread_info *curren
 #define TIF_BREAKPOINT  4 /* enter monitor mode on return */
 #define TIF_SINGLE_STEP  5 /* single step in progress */
 #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 */

 #define _TIF_SYSCALL_TRACE (1 << TIF_SYSCALL_TRACE)
 #define _TIF_SIGPENDING  (1 << TIF_SIGPENDING)
@@ -97,10 +98,11 @@ static inline struct thread_info *curren
 #define _TIF_POLLING_NRFLAG (1 << TIF_POLLING_NRFLAG)
 #define _TIF_SINGLE_STEP (1 << TIF_SINGLE_STEP)
 #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! */

 /* work to do on interrupt/exception return */
 #define _TIF_WORK_MASK    \
Index: linux-2.6.25-mm1/include/asm-cris/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-cris/thread_info.h
+++ linux-2.6.25-mm1/include/asm-cris/thread_info.h
@@ -84,17 +84,19 @@ struct thread_info {
 #define TIF_SIGPENDING  2 /* signal pending */
 #define TIF_NEED_RESCHED 3 /* rescheduling necessary */
 #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)
 #define _TIF_SIGPENDING  (1<<TIF_SIGPENDING)
 #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 */

 #endif /* __KERNEL__ */
Index: linux-2.6.25-mm1/include/asm-h8300/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-h8300/thread_info.h
+++ linux-2.6.25-mm1/include/asm-h8300/thread_info.h
@@ -90,17 +90,19 @@ static inline struct thread_info *curren
 #define TIF_NEED_RESCHED 2 /* rescheduling necessary */
 #define TIF_POLLING_NRFLAG 3 /* true if poll_idle() is polling
        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)
 #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)

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

 #endif /* __KERNEL__ */

Index: linux-2.6.25-mm1/include/asm-m68k/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-m68k/thread_info.h
+++ linux-2.6.25-mm1/include/asm-m68k/thread_info.h
@@ -56,7 +56,8 @@ struct thread_info {
 #define TIF_SIGPENDING  6 /* signal pending */
 #define TIF_NEED_RESCHED 7 /* rescheduling necessary */
 #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: linux-2.6.25-mm1/include/asm-m68knommu/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-m68knommu/thread_info.h
```

```
+++ linux-2.6.25-mm1/include/asm-m68knommu/thread_info.h
@@ -86,16 +86,18 @@ static inline struct thread_info *curren
 #define TIF_SIGPENDING  1 /* signal pending */
 #define TIF_NEED_RESCHED 2 /* rescheduling necessary */
 #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 */

 #endif /* __KERNEL__ */

Index: linux-2.6.25-mm1/include/asm-parisc/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-parisc/thread_info.h
+++ linux-2.6.25-mm1/include/asm-parisc/thread_info.h
@@ -60,17 +60,19 @@ struct thread_info {
 #define TIF_NEED_RESCHED 2 /* rescheduling necessary */
 #define TIF_POLLING_NRFLAG 3 /* true if poll_idle() is polling TIF_NEED_RESCHED */
 #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)
 #define _TIF_NEED_RESCHED (1 << TIF_NEED_RESCHED)
 #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)

 #endif /* __KERNEL__ */
Index: linux-2.6.25-mm1/include/asm-s390/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-s390/thread_info.h
+++ linux-2.6.25-mm1/include/asm-s390/thread_info.h
```

```
@@ -99,10 +99,11 @@ static inline struct thread_info *curren
 #define TIF_POLLING_NRFLAG 17 /* true if poll_idle() is polling
        TIF_NEED_RESCHED */
 #define TIF_31BIT  18 /* 32bit process */
 #define TIF_MEMDIE  19
 #define TIF_RESTORE_SIGMASK 20 /* restore signal mask in do_signal() */
+#define TIF_FREEZE  21 /* thread is freezing for suspend */

 #define _TIF_SYSCALL_TRACE (1<<TIF_SYSCALL_TRACE)
 #define _TIF_RESTORE_SIGMASK (1<<TIF_RESTORE_SIGMASK)
 #define _TIF_SIGPENDING  (1<<TIF_SIGPENDING)
 #define _TIF_NEED_RESCHED (1<<TIF_NEED_RESCHED)
@@ -111,10 +112,11 @@ static inline struct thread_info *curren
 #define _TIF_SINGLE_STEP (1<<TIF_SINGLE_STEP)
 #define _TIF_MCCK_PENDING (1<<TIF_MCCK_PENDING)
 #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__ */

 #define PREEMPT_ACTIVE  0x4000000

Index: linux-2.6.25-mm1/include/asm-sparc/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-sparc/thread_info.h
+++ linux-2.6.25-mm1/include/asm-sparc/thread_info.h
@@ -135,17 +135,19 @@ BTFIXUPDEF_CALL(void, free_thread_info,
 #define TIF_USEDFPU  8 /* FPU was used by this task
      * this quantum (SMP) */
 #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)
 #define _TIF_SIGPENDING  (1<<TIF_SIGPENDING)
 #define _TIF_NEED_RESCHED (1<<TIF_NEED_RESCHED)
 #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__ */

 #endif /* _ASM_THREAD_INFO_H */
```

```
Index: linux-2.6.25-mm1/include/asm-sparc64/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-sparc64/thread_info.h
+++ linux-2.6.25-mm1/include/asm-sparc64/thread_info.h
@@ -234,10 +234,11 @@ register struct thread_info *current_thr
  *       an immediate value in instructions such as andcc.
  */
 #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)
 #define _TIF_NEED_RESCHED (1<<TIF_NEED_RESCHED)
 #define _TIF_PERFCTR  (1<<TIF_PERFCTR)
@@ -247,10 +248,11 @@ register struct thread_info *current_thr
 #define _TIF_SECCOMP  (1<<TIF_SECCOMP)
 #define _TIF_SYSCALL_AUDIT (1<<TIF_SYSCALL_AUDIT)
 #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 | \
      _TIF_NEED_RESCHED | _TIF_PERFCTR))

Index: linux-2.6.25-mm1/include/asm-um/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-um/thread_info.h
+++ linux-2.6.25-mm1/include/asm-um/thread_info.h
@@ -81,15 +81,17 @@ static inline struct thread_info *curren
       */
 #define TIF_RESTART_BLOCK  4
 #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)
 #define _TIF_NEED_RESCHED (1 << TIF_NEED_RESCHED)
 #define _TIF_POLLING_NRFLAG     (1 << TIF_POLLING_NRFLAG)
 #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: linux-2.6.25-mm1/include/asm-v850/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-v850/thread_info.h
+++ linux-2.6.25-mm1/include/asm-v850/thread_info.h
@@ -80,16 +80,18 @@ struct thread_info {
 #define TIF_SIGPENDING  1 /* signal pending */
 #define TIF_NEED_RESCHED 2 /* rescheduling necessary */
 #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.  */
 #define THREAD_SIZE  0x2000

Index: linux-2.6.25-mm1/include/asm-xtensa/thread_info.h
===================================================================
--- linux-2.6.25-mm1.orig/include/asm-xtensa/thread_info.h
+++ linux-2.6.25-mm1/include/asm-xtensa/thread_info.h
@@ -136,18 +136,20 @@ static inline struct thread_info *curren
 #define TIF_SINGLESTEP  3 /* restore singlestep on return to user mode */
 #define TIF_IRET  4 /* return with iret */
 #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)
 #define _TIF_NEED_RESCHED (1<<TIF_NEED_RESCHED)
 #define _TIF_SINGLESTEP  (1<<TIF_SINGLESTEP)
 #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 2/5] Container Freezer: Make refrigerator always available

Posted by Matt Helsley on Thu, 24 Apr 2008 06:47:58 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>

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         |    4 -
 kernel/freezer.c        |  124 ++++++++++++++++++++++++++++++++++++++++++++++++
 kernel/power/process.c  |  113 ----------------------------------------
 4 files changed, 133 insertions(+), 127 deletions(-)

Index: linux-2.6.25-mm1/include/linux/freezer.h
===================================================================
--- linux-2.6.25-mm1.orig/include/linux/freezer.h
+++ linux-2.6.25-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-mm1/kernel/Makefile
===================================================================
--- linux-2.6.25-mm1.orig/kernel/Makefile
+++ linux-2.6.25-mm1/kernel/Makefile
@@ -3,15 +3,15 @@
 #

 obj-y     = sched.o fork.o exec_domain.o panic.o printk.o profile.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
+      notifier.o ksysfs.o pm_qos_params.o

 ifdef CONFIG_FTRACE
 # Do not profile debug utilities
 ORIG_CFLAGS := $(KBUILD_CFLAGS)
 KBUILD_CFLAGS = $(if $(filter-out lockdep% %debug,$(basename $(notdir $@))), \
Index: linux-2.6.25-mm1/kernel/freezer.c
===================================================================
--- /dev/null
+++ linux-2.6.25-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-mm1/kernel/power/process.c
===================================================================
--- linux-2.6.25-mm1.orig/kernel/power/process.c
+++ linux-2.6.25-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);

--
```

_____

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

---

Subject: [RFC][PATCH 3/5] Container Freezer: Implement freezer cgroup subsystem
Posted by Matt Helsley on Thu, 24 Apr 2008 06:47:59 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.

It's important to note that freezing can be incomplete. In that case we return
EBUSY. This means that some tasks in the cgroup are busy doing something that
prevents us from completely freezing the cgroup at this time. After EBUSY,
the cgroup will remain partially frozen -- reflected by freezer.state reporting
"FREEZING" when read. The state will remain "FREEZING" until one of these
things happens:

 1) Userspace cancels the freezing operation by writing "RUNNING" to
  the freezer.state file
 2) Userspace retries the freezing operation by writing "FROZEN" to
  the freezer.state file (writing "FREEZING" is not legal
  and returns EIO)
 3) The tasks that blocked the cgroup from entering the "FROZEN"
  state disappear from the cgroup's set of 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>

TODO:

Check that we handle ptrace'd and vfork-ing tasks correctly.

Changelog:
v2:
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

include/linux/cgroup_freezer.h | 65 +++++++++
include/linux/cgroup_subsys.h | 6
init/Kconfig | 7 +
kernel/Makefile | 1
kernel/cgroup_freezer.c | 276 ++
kernel/freezer.c | 1
6 files changed, 356 insertions(+)

Index: linux-2.6.25-mm1/include/linux/cgroup_freezer.h
===
--- /dev/null
+++ linux-2.6.25-mm1/include/linux/cgroup_freezer.h
@@ -0,0 +1,65 @@
+#ifndef _LINUX_CGROUP_FREEZER_H
+#define _LINUX_CGROUP_FREEZER_H
+/*
+ * cgroup_freezer.h - control group freezer subsystem interface
+ *
+ * Copyright IBM Corporation, 2007
+ *
+ * Author : Cedric Le Goater <clg@fr.ibm.com>
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2.1 of the GNU Lesser General Public License
+ * as published by the Free Software Foundation.
+ *
+ * This program is distributed in the hope that it would be useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
+ */

```
+
+#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-mm1/include/linux/cgroup_subsys.h
===================================================================
--- linux-2.6.25-mm1.orig/include/linux/cgroup_subsys.h
+++ linux-2.6.25-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-mm1/init/Kconfig
===================================================================
--- linux-2.6.25-mm1.orig/init/Kconfig
+++ linux-2.6.25-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-mm1/kernel/Makefile
===================================================================
--- linux-2.6.25-mm1.orig/kernel/Makefile
+++ linux-2.6.25-mm1/kernel/Makefile
@@ -46,10 +46,11 @@ 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-mm1/kernel/cgroup_freezer.c
===================================================================
--- /dev/null
+++ linux-2.6.25-mm1/kernel/cgroup_freezer.c
@@ -0,0 +1,276 @@
+/*
+ * cgroup_freezer.c -  control group freezer subsystem
+ *
+ * Copyright IBM Corporation, 2007
+ *
+ * Author : Cedric Le Goater <clg@fr.ibm.com>
+ *
+ * This program is free software; you can redistribute it and/or modify it
+ * under the terms of version 2.1 of the GNU Lesser General Public License
+ * as published by the Free Software Foundation.
+ *
+ * This program is distributed in the hope that it would be useful, but
+ * WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
+ */
+
+#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, ntotal = 0;
+
+ cgroup_iter_start(cgroup, &it);
+
+ while ((task = cgroup_iter_next(cgroup, &it))) {
+  ntotal++;
+  if (frozen(task))
+   nfrozen++;
+ }
+ cgroup_iter_end(cgroup, &it);
+
+ return (nfrozen == ntotal);
+}
+
+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_freeze_tasks(struct cgroup *cgroup)
+{
+ struct cgroup_iter it;
+ struct task_struct *task;
+ unsigned int num_cant_freeze_now = 0;
+
+ cgroup_iter_start(cgroup, &it);
+ while ((task = cgroup_iter_next(cgroup, &it))) {
+  if (!freeze_task(task, 1))
+   continue;
+  if (task_is_stopped_or_traced(task) && freezing(task))
+   /*
+    * So long as the freeze flag is set these tasks
```

```
+    * will immediately go into the fridge upon waking.
+    */
+   continue;
+  if (!freezing(task) && !freezer_should_skip(task))
+   num_cant_freeze_now++;
+ }
+ cgroup_iter_end(cgroup, &it);
+
+ return num_cant_freeze_now ? -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, enum freezer_state goal_state)
+{
+ struct freezer *freezer = cgroup_freezer(cgroup);
+ int retval = 0;
+
+ spin_lock_irq(&freezer->lock);
+retry:
+ if (goal_state == freezer->state)
+  goto unlock;
+
+ switch (freezer->state) {
+ case STATE_RUNNING:
+  if (goal_state == STATE_FROZEN) {
+   freezer->state = STATE_FREEZING;
+   retval = freezer_freeze_tasks(cgroup);
+  }
+  break;
+ case STATE_FREEZING:
+  if (freezer_check_if_frozen(cgroup)) {
+   freezer->state = STATE_FROZEN;
+   goto retry;
+  }
+
+  if (goal_state == STATE_FROZEN) {
```

```
+   /* Userspace is retrying after
+    * "echo FROZEN > freezer.state" returned -EBUSY */
+   retval = freezer_freeze_tasks(cgroup);
+   break;
+  }
+  /* goal_state == STATE_RUNNING, so unfreeze */
+ case STATE_FROZEN:
+  if (goal_state == STATE_RUNNING) {
+   freezer->state = STATE_RUNNING;
+   retval = freezer_unfreeze_tasks(cgroup);
+  }
+  break;
+ default:
+  break;
+ }
+unlock:
+ spin_unlock_irq(&freezer->lock);
+
+ return retval;
+}
+
+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)
+{
+ char *buffer;
+ int retval = 0;
+ enum freezer_state goal_state;
+
+ 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 */
+ strstrip(buffer);
+ if (strcmp(buffer, "RUNNING") == 0)
+  goal_state = STATE_RUNNING;
+ else if (strcmp(buffer, "FROZEN") == 0)
```

```
+  goal_state = STATE_FROZEN;
+ else {
+  retval = -EIO;
+  goto free_buffer;
+ }
+
+ cgroup_lock();
+
+ if (cgroup_is_removed(cgroup)) {
+  retval = -ENODEV;
+  goto unlock;
+ }
+
+ retval = freezer_freeze(cgroup, goal_state);
+ if (retval == 0)
+  retval = nbytes;
+unlock:
+ cgroup_unlock();
+free_buffer:
+ kfree(buffer);
+ return retval;
+}
+
+static struct cftype files[] = {
+ {
+  .name = "state",
+  .read = freezer_read,
+  .write = freezer_write,
+ },
+};
+
+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-mm1/kernel/freezer.c
```

```
===================================================================
--- linux-2.6.25-mm1.orig/kernel/freezer.c
+++ linux-2.6.25-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: [RFC][PATCH 4/5] Container Freezer: Skip frozen cgroups during power management resume

Posted by Matt Helsley on Thu, 24 Apr 2008 06:48:00 GMT
View Forum Message <> Reply to Message

When a system is resumed after a suspend, it will also unfreeze
frozen cgroups.

This patchs modifies the resume sequence to skip the tasks which
are part of a frozen control group.

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>
---
 kernel/power/process.c | 4 ++++
 1 file changed, 4 insertions(+)

Index: linux-2.6.25-mm1/kernel/power/process.c
```
===================================================================
--- linux-2.6.25-mm1.orig/kernel/power/process.c
+++ linux-2.6.25-mm1/kernel/power/process.c
@@ -11,10 +11,11 @@
 #include <linux/interrupt.h>
 #include <linux/suspend.h>
 #include <linux/module.h>
 #include <linux/syscalls.h>
 #include <linux/freezer.h>
+#include <linux/cgroup_freezer.h>
```

```
 /*
  * Timeout for stopping processes
  */
 #define TIMEOUT (20 * HZ)
@@ -150,10 +151,13 @@ static void thaw_tasks(int thaw_user_spa
    continue;

   if (!p->mm == thaw_user_space)
    continue;

+  if (cgroup_frozen(p))
+   continue;
+
   thaw_process(p);
  } while_each_thread(g, p);
  read_unlock(&tasklist_lock);
 }
```

--

_____

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

---

Subject: [RFC][PATCH 5/5] Add a Signal Control Group Subsystem
Posted by Matt Helsley on Thu, 24 Apr 2008 06:48:01 GMT
View Forum Message <> Reply to Message

Add a signal control group subsystem that allows us to send signals to all tasks
in the control group by writing the desired signal(7) number to the kill file.

NOTE: We don't really need per-cgroup state, but control groups doesn't support
stateless subsystems yet.

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

```
 include/linux/cgroup_signal.h |   28 +++++++++
 include/linux/cgroup_subsys.h |    6 +
 init/Kconfig                  |    6 +
 kernel/Makefile               |    1
 kernel/cgroup_signal.c        |  129 ++++++++++++++++++++++++++++++++++++++++++++
 5 files changed, 170 insertions(+)
```

Index: linux-2.6.25-mm1/include/linux/cgroup_signal.h
===
--- /dev/null

```
+++ linux-2.6.25-mm1/include/linux/cgroup_signal.h
@@ -0,0 +1,28 @@
+#ifndef _LINUX_CGROUP_SIGNAL_H
+#define _LINUX_CGROUP_SIGNAL_H
+/*
+ * cgroup_signal.h -  control group freezer subsystem interface
+ *
+ * Copyright IBM Corp. 2007
+ *
+ * Author : Cedric Le Goater <clg@fr.ibm.com>
+ * Author : Matt Helsley <matthltc@us.ibm.com>
+ */
+
+#include <linux/cgroup.h>
+
+#ifdef CONFIG_CGROUP_SIGNAL
+
+struct stateless {
+ struct cgroup_subsys_state css;
+};
+
+static inline struct stateless *cgroup_signal(struct cgroup *cgroup)
+{
+ return container_of(cgroup_subsys_state(cgroup, signal_subsys_id),
+       struct stateless, css);
+}
+
+#else /* !CONFIG_CGROUP_SIGNAL */
+#endif /* !CONFIG_CGROUP_SIGNAL */
+#endif /* _LINUX_CGROUP_SIGNAL_H */
Index: linux-2.6.25-mm1/kernel/cgroup_signal.c
===================================================================
--- /dev/null
+++ linux-2.6.25-mm1/kernel/cgroup_signal.c
@@ -0,0 +1,129 @@
+/*
+ * cgroup_signal.c -  control group signal subsystem
+ *
+ * Copyright IBM Corp. 2007
+ *
+ * Author : Cedric Le Goater <clg@fr.ibm.com>
+ * Author : Matt Helsley <matthltc@us.ibm.com>
+ */
+
+#include <linux/module.h>
+#include <linux/cgroup.h>
+#include <linux/fs.h>
+#include <linux/uaccess.h>
```

```
+#include <linux/cgroup_signal.h>
+
+struct cgroup_subsys signal_subsys;
+
+static struct cgroup_subsys_state *signal_create(
+ struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ struct stateless *dummy;
+
+ if (!capable(CAP_SYS_ADMIN))
+  return ERR_PTR(-EPERM);
+
+ dummy = kzalloc(sizeof(struct stateless), GFP_KERNEL);
+ if (!dummy)
+  return ERR_PTR(-ENOMEM);
+ return  &dummy->css;
+}
+
+static void signal_destroy(struct cgroup_subsys *ss,
+       struct cgroup *cgroup)
+{
+ kfree(cgroup_signal(cgroup));
+}
+
+
+static int signal_can_attach(struct cgroup_subsys *ss,
+        struct cgroup *new_cgroup,
+        struct task_struct *task)
+{
+ return 0;
+}
+
+static int signal_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 ssize_t signal_write(struct cgroup *cgroup,
+        struct cftype *cft,
+        struct file *file,
+        const char __user *userbuf,
+        size_t nbytes, loff_t *unused_ppos)
+{
+ 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 */
+ if (sscanf(buffer, "%d", &value) != 1) {
+  retval = -EIO;
+  goto free_buffer;
+ }
+
+ cgroup_lock();
+
+ if (cgroup_is_removed(cgroup)) {
+  retval = -ENODEV;
+  goto unlock;
+ }
+
+ retval = signal_kill(cgroup, value);
+ if (retval == 0)
+  retval = nbytes;
+unlock:
+ cgroup_unlock();
+free_buffer:
+ kfree(buffer);
+ return retval;
+}
+
+static struct cftype kill_file = {
+ .name = "kill",
```

```
+ .write = signal_write,
+ .private = 0,
+};
+
+static int signal_populate(struct cgroup_subsys *ss, struct cgroup *cgroup)
+{
+ return cgroup_add_files(cgroup, ss, &kill_file, 1);
+}
+
+struct cgroup_subsys signal_subsys = {
+ .name  = "signal",
+ .create  = signal_create,
+ .destroy = signal_destroy,
+ .populate = signal_populate,
+ .subsys_id = signal_subsys_id,
+ .can_attach = signal_can_attach,
+ .attach  = NULL,
+ .fork  = NULL,
+ .exit  = NULL,
+};
Index: linux-2.6.25-mm1/init/Kconfig
===================================================================
--- linux-2.6.25-mm1.orig/init/Kconfig
+++ linux-2.6.25-mm1/init/Kconfig
@@ -328,10 +328,16 @@ config CGROUP_FREEZER
         depends on CGROUPS
         help
           Provides a way to freeze and unfreeze all tasks in a
    cgroup

+config CGROUP_SIGNAL
+        bool "control group signal subsystem"
+        depends on CGROUPS
+        help
+          Provides a way to signal 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-mm1/kernel/Makefile
===================================================================
--- linux-2.6.25-mm1.orig/kernel/Makefile
+++ linux-2.6.25-mm1/kernel/Makefile
@@ -47,10 +47,11 @@ 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_CGROUP_SIGNAL) += cgroup_signal.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-mm1/include/linux/cgroup_subsys.h
===================================================================
--- linux-2.6.25-mm1.orig/include/linux/cgroup_subsys.h
+++ linux-2.6.25-mm1/include/linux/cgroup_subsys.h
@@ -52,5 +52,11 @@ SUBSYS(devices)
 #ifdef CONFIG_CGROUP_FREEZER
 SUBSYS(freezer)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_SIGNAL
+SUBSYS(signal)
+#endif
+
+/* */

--
```

_____

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

---

Subject: Re: [RFC][PATCH 1/5] Container Freezer: Add TIF_FREEZE flag to all architectures

Posted by Pavel Machek on Thu, 24 Apr 2008 08:09:18 GMT
View Forum Message <> Reply to Message

On Wed 2008-04-23 23:47:57, Matt Helsley 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>

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 5/5] Add a Signal Control Group Subsystem
Posted by Paul Jackson on Thu, 24 Apr 2008 19:30:32 GMT
View Forum Message <> Reply to Message

> +static struct cftype kill_file = {
> + .name = "kill",

The name "kill" seems ambiguous to me. It suggests that any write
will send some default signal (TERM or KILL?) to all tasks in the
cgroup, rather like the 'killall' command.

I'm guessing that more people, on seeing this file in a cgroup
directory, will guess correctly what it does if it were named
"signal" or "send_signal" or some such.

--
I won't rest till it's the best ...
Programmer, Linux Scalability
Paul Jackson <pj@sgi.com> 1.940.382.4214

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

---

Subject: Re: [RFC][PATCH 3/5] Container Freezer: Implement freezer cgroup subsystem
Posted by Paul Menage on Fri, 25 Apr 2008 05:51:57 GMT
View Forum Message <> Reply to Message

>+static const char *freezer_state_strs[] = {
>+ "RUNNING\n",
>+ "FREEZING\n" ,
>+ "FROZEN\n"

>+};

I think it might be cleaner to not include the \n characters in this array.

>+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);
>+}

You need to be in an RCU critical section or else hold task_lock() in
order to dereference the cgroup returned from task_cgroup()

I'm not sure that you need to take freezer->lock here - you're just
reading a single word.

>+
>+ if (!capable(CAP_SYS_ADMIN))
>+ return ERR_PTR(-EPERM);
>+

Why does everyone keep throwing calls to check CAP_SYS_ADMIN into
their cgroup create callbacks? You have to be root in order to mount a
cgroups hierarchy in the first place, and filesystem permissions will
control who can create new cgroups.

>+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;
>+}

You should comment here that 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 ssize_t freezer_write(struct cgroup *cgroup,
>+        struct cftype *cft,
>+        struct file *file,
>+        const char __user *userbuf,
>+        size_t nbytes, loff_t *unused_ppos)
>+{
>+ char *buffer;
>+ int retval = 0;
>+ enum freezer_state goal_state;
>+
>+ if (nbytes >= PATH_MAX)
>+  return -E2BIG;
>+
>+ /* +1 for nul-terminator */
>+ buffer = kmalloc(nbytes + 1, GFP_KERNEL);
>+ if (buffer == NULL)
>+  return -ENOMEM;
```

Given that you're copying a string whose maximum valid length is
"FREEZING" you don't really need to use a dynamically-allocated
buffer.

But I really ought to provide a write_string() method that handles
this kind of copying on behalf of cgroup subsystems, the way it
already does for 64-bit ints.

```
>+ if (strcmp(buffer, "RUNNING") == 0)
>+  goal_state = STATE_RUNNING;
>+ else if (strcmp(buffer, "FROZEN") == 0)
>+  goal_state = STATE_FROZEN;
```

Would it make sense to compare against the strings you already have in
the array earlier in the file?

Paul

Subject: Re: [RFC][PATCH 5/5] Add a Signal Control Group Subsystem
Posted by Paul Menage on Fri, 25 Apr 2008 06:01:46 GMT
View Forum Message <> Reply to Message

I don't think you need cgroup_signal.h. It's only included in
cgroup_signal.c, and doesn't really contain any useful definitions
anyway. You should just use a cgroup_subsys_state object as your state
object, since you'll never need to do anything with it anyway.

```
>+static struct cgroup_subsys_state *signal_create(
>+ struct cgroup_subsys *ss, struct cgroup *cgroup)
>+{
>+ struct stateless *dummy;
>+
>+ if (!capable(CAP_SYS_ADMIN))
>+  return ERR_PTR(-EPERM);
```

This is unnecessary.

```
>+
+ dummy = kzalloc(sizeof(struct stateless), GFP_KERNEL);
+ if (!dummy)
+  return ERR_PTR(-ENOMEM);
+ return  &dummy->css;
+}
```

This function could be simplified to:

```
struct cgroup_subsys_state *css;
css = kzalloc(sizeof(*css), GFP_KERNEL);
return css ?: ERR_PTR(-ENOMEM);
```

```
>+static int signal_can_attach(struct cgroup_subsys *ss,
>+        struct cgroup *new_cgroup,
>+        struct task_struct *task)
>+{
>+ return 0;
>+}
```

No need for a can_attach() method if it just returns 0 - that's the default.

```
>+static int signal_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;
>+}

cgroup_iter_start() takes a read lock - is send_sig() guaranteed not to sleep?

>+static ssize_t signal_write(struct cgroup *cgroup,
>+ struct cftype *cft,
>+ struct file *file,
>+ const char __user *userbuf,
>+ size_t nbytes, loff_t *unused_ppos)

This should just be a write_u64() method - cgroups will handle the
copying/parsing for you. See e.g.
kernel/sched.c:cpu_shares_write_u64()

>+static struct cftype kill_file = {
>+ .name = "kill",
>+ .write = signal_write,
>+ .private = 0,
>+};

I agree with PaulJ that "signal.send" would be a nicer name for this
than "signal.kill"

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

---

Subject: Re: [RFC][PATCH 2/5] Container Freezer: Make refrigerator always available
Posted by Pavel Machek on Fri, 25 Apr 2008 11:04:56 GMT
View Forum Message <> Reply to Message

Hi!

> 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>

There's no problem with doing this... but you should get some debate
(with Linus?) whether using freezer for cgroups is sane. When that is
done, there's no problem with this going in, probably through rafael's
patch queue.

(The first patch -- add freezer for all archs -- is probably
reasonably to go in ASAP, through akpm or something...)
	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 5/5] Add a Signal Control Group Subsystem
Posted by Cedric Le Goater on Fri, 25 Apr 2008 11:41:49 GMT
View Forum Message <> Reply to Message

Matt Helsley wrote:
> Add a signal control group subsystem that allows us to send signals to all tasks
> in the control group by writing the desired signal(7) number to the kill file.
>
> NOTE: We don't really need per-cgroup state, but control groups doesn't support
> stateless subsystems yet.
>
> Signed-off-by: Matt Helsley <matthltc@us.ibm.com>
> ---
> include/linux/cgroup_signal.h | 28 +++++++++
> include/linux/cgroup_subsys.h | 6 +
> init/Kconfig | 6 +
> kernel/Makefile | 1
> kernel/cgroup_signal.c | 129 ++
> 5 files changed, 170 insertions(+)
>
> Index: linux-2.6.25-mm1/include/linux/cgroup_signal.h
> ===
> --- /dev/null
> +++ linux-2.6.25-mm1/include/linux/cgroup_signal.h
> @@ -0,0 +1,28 @@
> +#ifndef _LINUX_CGROUP_SIGNAL_H
> +#define _LINUX_CGROUP_SIGNAL_H
> +/*
> + * cgroup_signal.h - control group freezer subsystem interface

s/freezer/signal/

> + *
> + * Copyright IBM Corp. 2007
> + *
> + * Author : Cedric Le Goater <clg@fr.ibm.com>
> + * Author : Matt Helsley <matthltc@us.ibm.com>
> + */
> +
> +#include <linux/cgroup.h>
> +
> +#ifdef CONFIG_CGROUP_SIGNAL
> +
> +struct stateless {
> + struct cgroup_subsys_state css;
> +};

I'm not sure this is correct to say so. Imagine you want to send
a SIGKILL to a cgroup, you would expect all tasks to die and the
cgroup to become empty. right ?

but if a task is doing clone() while it's being killed by this cgroup
signal subsystem, we can miss the child. This is because there's a
small window in copy_process() where the child is in the cgroup and
not visible yet.

```
    copy_process()
 cgroup_fork()
  do stuff
 cgroup_fork_callbacks()

 cgroup_post_fork()
  put new task in the list.
```

(I didn't dig too much the code, though. So I might be missing
  something)

So if we want to send the signal to all tasks in the cgroup, we need
to track the new tasks with a fork callback, just like the freezer :

```
static void signal_fork(struct cgroup_subsys *ss, struct task_struct *task)
{

}
```

and, of course, we need to keep somewhere the signal number we need to
send.

All this depends on how we want the cgroup signal subsystem to behave.
It could be brainless of course, but it seems to me that the biggest
benefit of such a subsystem is to use the cgroup capability to track
new tasks coming in.

Cheers,

C.

> +static inline struct stateless *cgroup_signal(struct cgroup *cgroup)
> +{
> + return container_of(cgroup_subsys_state(cgroup, signal_subsys_id),
> + struct stateless, css);
> +}
> +
> +#else /* !CONFIG_CGROUP_SIGNAL */
> +#endif /* !CONFIG_CGROUP_SIGNAL */
> +#endif /* _LINUX_CGROUP_SIGNAL_H */
> Index: linux-2.6.25-mm1/kernel/cgroup_signal.c
> ===
> --- /dev/null
> +++ linux-2.6.25-mm1/kernel/cgroup_signal.c
> @@ -0,0 +1,129 @@
> +/*
> + * cgroup_signal.c - control group signal subsystem
> + *
> + * Copyright IBM Corp. 2007
> + *
> + * Author : Cedric Le Goater <clg@fr.ibm.com>
> + * Author : Matt Helsley <matthltc@us.ibm.com>
> + */
> +
> +#include <linux/module.h>
> +#include <linux/cgroup.h>
> +#include <linux/fs.h>
> +#include <linux/uaccess.h>
> +#include <linux/cgroup_signal.h>
> +
> +struct cgroup_subsys signal_subsys;
> +
> +static struct cgroup_subsys_state *signal_create(
> + struct cgroup_subsys *ss, struct cgroup *cgroup)
> +{
> + struct stateless *dummy;
> +
> + if (!capable(CAP_SYS_ADMIN))

```
> +  return ERR_PTR(-EPERM);
> +
> + dummy = kzalloc(sizeof(struct stateless), GFP_KERNEL);
> + if (!dummy)
> +  return ERR_PTR(-ENOMEM);
> + return  &dummy->css;
> +}
> +
> +static void signal_destroy(struct cgroup_subsys *ss,
> +       struct cgroup *cgroup)
> +{
> + kfree(cgroup_signal(cgroup));
> +}
> +
> +
> +static int signal_can_attach(struct cgroup_subsys *ss,
> +         struct cgroup *new_cgroup,
> +         struct task_struct *task)
> +{
> + return 0;
> +}
> +
> +static int signal_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 ssize_t signal_write(struct cgroup *cgroup,
> +         struct cftype *cft,
> +         struct file *file,
> +         const char __user *userbuf,
> +         size_t nbytes, loff_t *unused_ppos)
> +{
> + 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 */
> + if (sscanf(buffer, "%d", &value) != 1) {
> +  retval = -EIO;
> +  goto free_buffer;
> + }
> +
> + cgroup_lock();
> +
> + if (cgroup_is_removed(cgroup)) {
> +  retval = -ENODEV;
> +  goto unlock;
> + }
> +
> + retval = signal_kill(cgroup, value);
> + if (retval == 0)
> +  retval = nbytes;
> +unlock:
> + cgroup_unlock();
> +free_buffer:
> + kfree(buffer);
> + return retval;
> +}
> +
> +static struct cftype kill_file = {
> + .name = "kill",
> + .write = signal_write,
> + .private = 0,
> +};
> +
> +static int signal_populate(struct cgroup_subsys *ss, struct cgroup *cgroup)
> +{
> + return cgroup_add_files(cgroup, ss, &kill_file, 1);
> +}
> +
> +struct cgroup_subsys signal_subsys = {
```

> + .name = "signal",
> + .create = signal_create,
> + .destroy = signal_destroy,
> + .populate = signal_populate,
> + .subsys_id = signal_subsys_id,
> + .can_attach = signal_can_attach,
> + .attach = NULL,
> + .fork = NULL,
> + .exit = NULL,
> +};
> Index: linux-2.6.25-mm1/init/Kconfig
> ===
> --- linux-2.6.25-mm1.orig/init/Kconfig
> +++ linux-2.6.25-mm1/init/Kconfig
> @@ -328,10 +328,16 @@ config CGROUP_FREEZER
> depends on CGROUPS
> help
> Provides a way to freeze and unfreeze all tasks in a
> cgroup
>
> +config CGROUP_SIGNAL
> + bool "control group signal subsystem"
> + depends on CGROUPS
> + help
> + Provides a way to signal 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-mm1/kernel/Makefile
> ===
> --- linux-2.6.25-mm1.orig/kernel/Makefile
> +++ linux-2.6.25-mm1/kernel/Makefile
> @@ -47,10 +47,11 @@ 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-$(CONFIG_CGROUP_SIGNAL) += cgroup_signal.o
> obj-$(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-mm1/include/linux/cgroup_subsys.h

> ===
> --- linux-2.6.25-mm1.orig/include/linux/cgroup_subsys.h
> +++ linux-2.6.25-mm1/include/linux/cgroup_subsys.h
> @@ -52,5 +52,11 @@ SUBSYS(devices)
> #ifdef CONFIG_CGROUP_FREEZER
> SUBSYS(freezer)
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CGROUP_SIGNAL
> +SUBSYS(signal)
> +#endif
> +
> +/* */
>

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

---

Subject: Re: [RFC][PATCH 2/5] Container Freezer: Make refrigerator always available
Posted by Cedric Le Goater on Fri, 25 Apr 2008 12:07:20 GMT
View Forum Message <> Reply to Message

Pavel Machek wrote:
> Hi!
>
>> 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>
>
> There's no problem with doing this... but you should get some debate
> (with Linus?) whether using freezer for cgroups is sane.

Yes that's what we are trying to know, is the fake signal mechanism
used by the freezer something we can build upon ?

> When that is done, there's no problem with this going in, probably
> through rafael's patch queue.

OK

> (The first patch -- add freezer for all archs -- is probably
> reasonably to go in ASAP, through akpm or something...)

Thanks,

C.

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

---

Subject: Re: [RFC][PATCH 2/5] Container Freezer: Make refrigerator always available
Posted by rjw on Sat, 26 Apr 2008 13:02:36 GMT
View Forum Message <> Reply to Message

On Friday, 25 of April 2008, Cedric Le Goater wrote:
> Pavel Machek wrote:
> > Hi!
> >
> >> 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>
> >
> > There's no problem with doing this... but you should get some debate
> > (with Linus?) whether using freezer for cgroups is sane.
>
> Yes that's what we are trying to know, is the fake signal mechanism
> used by the freezer something we can build upon ?

Well, currently, the freezer doesn't send fake signals to kernel threads which
turns out to be problematic, becase some kernel threads behave very much like
user space processes (basically, the threads related to NFS and CIFS).

I have an idea how to fix this issue, but I'm not sure if it's acceptable

overall. I'll try to prepare a patch later today or tomorrow.

Thanks,
Rafael

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

---

Subject: [RFC][PATCH] Freezer: NOSIG flag (was: Re: [RFC][PATCH 2/5] Container Freezer: Make refrigerator alw
Posted by rjw on Sat, 26 Apr 2008 23:32:40 GMT
View Forum Message <> Reply to Message

On Saturday, 26 of April 2008, Rafael J. Wysocki wrote:
> On Friday, 25 of April 2008, Cedric Le Goater wrote:
> > Pavel Machek wrote:
> > > Hi!
> > >
> > >> 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>
> > >
> > > There's no problem with doing this... but you should get some debate
> > > (with Linus?) whether using freezer for cgroups is sane.
> >
> > Yes that's what we are trying to know, is the fake signal mechanism
> > used by the freezer something we can build upon ?
>
> Well, currently, the freezer doesn't send fake signals to kernel threads which
> turns out to be problematic, becase some kernel threads behave very much like
> user space processes (basically, the threads related to NFS and CIFS).
>
> I have an idea how to fix this issue, but I'm not sure if it's acceptable
> overall. I'll try to prepare a patch later today or tomorrow.

Below is the patch I was talking about, the changelog says what it does.

I have some candidates for using set_freezable_with_signal(), but I need to
check the test cases before modifying them.

Comments welcome.

Thanks,
Rafael

From: Rafael J. Wysocki <rjw@sisk.pl>

The freezer currently attempts to distinguish kernel threads from
user space tasks by checking if their mm pointer is unset and it
does not send fake signals to kernel threads. However, there are
kernel threads, mostly related to networking, that behave like
user space tasks and may want to be sent a fake signal to be frozen.

Introduce the new process flag PF_FREEZER_NOSIG that will be set
by default for all kernel threads and make the freezer only send
fake signals to the tasks having PF_FREEZER_NOSIG unset. Provide
the set_freezable_with_signal() function to be called by the kerlel
threads that want to be sent a fake signal for freezing.

This patch should not change the freezer's observable behavior.

Signed-off-by: Rafael J. Wysocki <rjw@sisk.pl>
---
 include/linux/freezer.h | 10 ++++
 include/linux/sched.h | 1
 kernel/kthread.c | 2
 kernel/power/process.c | 97 ++++++++++++++++++++++++++----------------------
 4 files changed, 54 insertions(+), 56 deletions(-)

Index: linux-2.6/include/linux/freezer.h
===
--- linux-2.6.orig/include/linux/freezer.h
+++ linux-2.6/include/linux/freezer.h
@@ -128,6 +128,15 @@ static inline void set_freezable(void)
 }

 /*
+ * Tell the freezer that the current task should be frozen by it and that it
+ * should send a fake signal to the task to freeze it.
+ */
+static inline void set_freezable_with_signal(void)
+{
+ current->flags &= ~(PF_NOFREEZE | PF_FREEZER_NOSIG);
+}
+
+/*

```
  * Freezer-friendly wrappers around wait_event_interruptible() and
  * wait_event_interruptible_timeout(), originally defined in <linux/wait.h>
  */
@@ -174,6 +183,7 @@ static inline void freezer_do_not_count(
 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) {}

 #define wait_event_freezable(wq, condition)    \
   wait_event_interruptible(wq, condition)
Index: linux-2.6/include/linux/sched.h
===================================================================
--- linux-2.6.orig/include/linux/sched.h
+++ linux-2.6/include/linux/sched.h
@@ -1499,6 +1499,7 @@ static inline void put_task_struc
 #define PF_MEMPOLICY 0x10000000 /* Non-default NUMA mempolicy */
 #define PF_MUTEX_TESTER 0x20000000 /* Thread belongs to the rt mutex tester */
 #define PF_FREEZER_SKIP 0x40000000 /* Freezer should not count it as freezeable */
+#define PF_FREEZER_NOSIG 0x80000000 /* Freezer won't send signals to it */

 /*
  * Only the _current_ task can read/write to tsk->flags, but other
Index: linux-2.6/kernel/power/process.c
===================================================================
--- linux-2.6.orig/kernel/power/process.c
+++ linux-2.6/kernel/power/process.c
@@ -19,9 +19,6 @@
  */
 #define TIMEOUT (20 * HZ)

-#define FREEZER_KERNEL_THREADS 0
-#define FREEZER_USER_SPACE 1
-
 static inline int freezeable(struct task_struct * p)
 {
  if ((p == current) ||
@@ -84,63 +81,53 @@ static void fake_signal_wake_up(struct t
  spin_unlock_irqrestore(&p->sighand->siglock, flags);
 }

-static int has_mm(struct task_struct *p)
+static inline bool should_send_signal(struct task_struct *p)
 {
- return (p->mm && !(p->flags & PF_BORROWED_MM));
+ return !(current->flags & PF_FREEZER_NOSIG);
 }
```

```
 /**
  * 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
+ * @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 seting the tasks's TIF_FREEZE flag and
+ * 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 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.
+ * 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 int freeze_task(struct task_struct *p, int with_mm_only)
+static bool freeze_task(struct task_struct *p, bool sig_only)
 {
- int ret = 1;
+ /*
+  * 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;

- 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);
-  }
+  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 {
-  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);
-      }
-    }
-  }
+  wake_up_state(p, TASK_INTERRUPTIBLE);
   }
- task_unlock(p);
- return ret;
+
+ return true;
 }

 static void cancel_freezing(struct task_struct *p)
@@ -156,7 +143,7 @@ static void cancel_freezing(struct task_
   }
 }

-static int try_to_freeze_tasks(int freeze_user_space)
+static int try_to_freeze_tasks(bool sig_only)
 {
   struct task_struct *g, *p;
   unsigned long end_time;
```

```
@@ -175,7 +162,7 @@ static int try_to_freeze_tasks(int freez
    if (frozen(p) || !freezeable(p))
     continue;

-   if (!freeze_task(p, freeze_user_space))
+   if (!freeze_task(p, sig_only))
     continue;

    /*
@@ -235,13 +222,13 @@ int freeze_processes(void)
  int error;

  printk("Freezing user space processes ... ");
- error = try_to_freeze_tasks(FREEZER_USER_SPACE);
+ error = try_to_freeze_tasks(true);
  if (error)
   goto Exit;
  printk("done.\n");

  printk("Freezing remaining freezable tasks ... ");
- error = try_to_freeze_tasks(FREEZER_KERNEL_THREADS);
+ error = try_to_freeze_tasks(false);
  if (error)
   goto Exit;
  printk("done.");
@@ -251,7 +238,7 @@ int freeze_processes(void)
  return error;
 }

-static void thaw_tasks(int thaw_user_space)
+static void thaw_tasks(bool sig_only)
 {
  struct task_struct *g, *p;

@@ -260,7 +247,7 @@ static void thaw_tasks(int thaw_user_spa
   if (!freezeable(p))
    continue;

-  if (!p->mm == thaw_user_space)
+  if (sig_only && !should_send_signal(p))
    continue;

   thaw_process(p);
@@ -271,8 +258,8 @@ static void thaw_tasks(int thaw_user_spa
 void thaw_processes(void)
 {
  printk("Restarting tasks ... ");
- thaw_tasks(FREEZER_KERNEL_THREADS);
```

```
- thaw_tasks(FREEZER_USER_SPACE);
+ thaw_tasks(true);
+ thaw_tasks(false);
  schedule();
  printk("done.\n");
 }
Index: linux-2.6/kernel/kthread.c
===================================================================
--- linux-2.6.orig/kernel/kthread.c
+++ linux-2.6/kernel/kthread.c
@@ -234,7 +234,7 @@ int kthreadd(void *unused)
  set_user_nice(tsk, KTHREAD_NICE_LEVEL);
  set_cpus_allowed(tsk, CPU_MASK_ALL);

- current->flags |= PF_NOFREEZE;
+ current->flags |= PF_NOFREEZE | PF_FREEZER_NOSIG;

  for (;;) {
   set_current_state(TASK_INTERRUPTIBLE);
```

Subject: Re: [RFC][PATCH 3/5] Container Freezer: Implement freezer cgroup subsystem

Posted by serue on Mon, 28 Apr 2008 04:03:58 GMT

View Forum Message <> Reply to Message

Quoting Paul Menage (menage@google.com):

```
> >+static const char *freezer_state_strs[] = {
> >+ "RUNNING\n",
> >+ "FREEZING\n" ,
> >+ "FROZEN\n"
> >+};
>
> I think it might be cleaner to not include the \n characters in this array.
>
> >+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);
> >+}
>
> You need to be in an RCU critical section or else hold task_lock() in
> order to dereference the cgroup returned from task_cgroup()
>
> I'm not sure that you need to take freezer->lock here - you're just
> reading a single word.
>
> >+
> >+ if (!capable(CAP_SYS_ADMIN))
> >+ return ERR_PTR(-EPERM);
> >+
>
> Why does everyone keep throwing calls to check CAP_SYS_ADMIN into
> their cgroup create callbacks? You have to be root in order to mount a
> cgroups hierarchy in the first place, and filesystem permissions will
> control who can create new cgroups.

The scourge of cut-n-paste :) Except I'm thinking that the check should
be taken out of even kernel/ns_cgroup.c:ns_create(), which I think is
where that all began.

The reason why tossing these in is bad is that it requires us to give
away extra privilege.

> >+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;
> >+}
>
> You should comment here that 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 ssize_t freezer_write(struct cgroup *cgroup,
> >+   struct cftype *cft,
> >+   struct file *file,

> >+ const char __user *userbuf,
> >+ size_t nbytes, loff_t *unused_ppos)
> >+{
> >+ char *buffer;
> >+ int retval = 0;
> >+ enum freezer_state goal_state;
> >+
> >+ if (nbytes >= PATH_MAX)
> >+ return -E2BIG;
> >+
> >+ /* +1 for nul-terminator */
> >+ buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> >+ if (buffer == NULL)
> >+ return -ENOMEM;
>
> Given that you're copying a string whose maximum valid length is
> "FREEZING" you don't really need to use a dynamically-allocated
> buffer.
>
> But I really ought to provide a write_string() method that handles
> this kind of copying on behalf of cgroup subsystems, the way it
> already does for 64-bit ints.
>
> >+ if (strcmp(buffer, "RUNNING") == 0)
> >+ goal_state = STATE_RUNNING;
> >+ else if (strcmp(buffer, "FROZEN") == 0)
> >+ goal_state = STATE_FROZEN;
>
> Would it make sense to compare against the strings you already have in
> the array earlier in the file?
>
> Paul
> _____
> 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 5/5] Add a Signal Control Group Subsystem
Posted by Matt Helsley on Wed, 30 Apr 2008 07:48:34 GMT
View Forum Message <> Reply to Message

On Thu, 2008-04-24 at 14:30 -0500, Paul Jackson wrote:
> > +static struct cftype kill_file = {

> > + .name = "kill",
>
> The name "kill" seems ambiguous to me. It suggests that any write
> will send some default signal (TERM or KILL?) to all tasks in the
> cgroup, rather like the 'killall' command.
>
> I'm guessing that more people, on seeing this file in a cgroup
> directory, will guess correctly what it does if it were named
> "signal" or "send_signal" or some such.

OK, I renamed it signal.send and replaced all uses of "kill" in the code
to indicate that we're actually sending a signal -- not "KILL"ing
things :).

Cheers,
 -Matt

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

---

Subject: Re: [RFC][PATCH 5/5] Add a Signal Control Group Subsystem
Posted by Paul Jackson on Wed, 30 Apr 2008 08:18:57 GMT
View Forum Message <> Reply to Message

Matt wrote:
> OK, I renamed it signal.send

I'm not familiar with the cgroup subsystem naming conventions,
but modulo that, this looks good to me - thanks!

--
                  I won't rest till it's the best ...
                  Programmer, Linux Scalability
                  Paul Jackson <pj@sgi.com> 1.940.382.4214

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

---

Subject: Re: [RFC][PATCH 5/5] Add a Signal Control Group Subsystem
Posted by Matt Helsley on Wed, 30 Apr 2008 08:29:08 GMT
View Forum Message <> Reply to Message

On Thu, 2008-04-24 at 23:01 -0700, Paul Menage wrote:
> I don't think you need cgroup_signal.h. It's only included in
> cgroup_signal.c, and doesn't really contain any useful definitions
> anyway. You should just use a cgroup_subsys_state object as your state
> object, since you'll never need to do anything with it anyway.
>
> >+static struct cgroup_subsys_state *signal_create(
> >+ struct cgroup_subsys *ss, struct cgroup *cgroup)
> >+{
> >+ struct stateless *dummy;
> >+
> >+ if (!capable(CAP_SYS_ADMIN))
> >+ return ERR_PTR(-EPERM);
>
> This is unnecessary.

OK, removed.

> >+
> + dummy = kzalloc(sizeof(struct stateless), GFP_KERNEL);
> + if (!dummy)
> + return ERR_PTR(-ENOMEM);
> + return &dummy->css;
> +}
>
> This function could be simplified to:
>
> struct cgroup_subsys_state *css;
> css = kzalloc(sizeof(*css), GFP_KERNEL);
> return css ?: ERR_PTR(-ENOMEM);

I kept the if() syntax but used cgroup_subsys_state as suggested. I kept the name "dummy" too to emphasize that except for following the currently-required form we don't really need the state information.

As a side note, I don't think adding state (which signal was/to sent/send) or a fork handling function prevents races. I tend to agree that there are lots of races involving adding/removing tasks to the cgroup where we may not signal "everything". I think that from userspace's perspective we can't solve the races because there will always be a window between releasing whatever lock we use and returning to userspace where new tasks can be added.

IMHO the only way to prevent such races would be to allow userspace to "lock" a cgroup so that no new tasks may be added. That would block/fail new forks and prevent writing new pids to the tasks file. Otherwise userspace must always recheck the list to ensure it didn't get any new entries...

> >+static int signal_can_attach(struct cgroup_subsys *ss,
> >+ struct cgroup *new_cgroup,
> >+ struct task_struct *task)
> >+{
> >+ return 0;
> >+}
>
> No need for a can_attach() method if it just returns 0 - that's the default.

Removed.

> >+static int signal_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;
> >+}
>
> cgroup_iter_start() takes a read lock - is send_sig() guaranteed not to sleep?

send_sig() -> send_sig_info() hold the tasklist_lock for read and the
task's sighand->siglock spinlock.

> >+static ssize_t signal_write(struct cgroup *cgroup,
> >+ struct cftype *cft,
> >+ struct file *file,
> >+ const char __user *userbuf,
> >+ size_t nbytes, loff_t *unused_ppos)
>
> This should just be a write_u64() method - cgroups will handle the
> copying/parsing for you. See e.g.
> kernel/sched.c:cpu_shares_write_u64()

Sure.

> >+static struct cftype kill_file = {
> >+ .name = "kill",

> >+ .write = signal_write,
> >+ .private = 0,
> >+};
>
> I agree with PaulJ that "signal.send" would be a nicer name for this
> than "signal.kill"

OK.

Cheers,
 -Matt Helsley

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

---

Subject: Re: [RFC][PATCH 2/5] Container Freezer: Make refrigerator always available
Posted by Matt Helsley on Wed, 30 Apr 2008 09:08:29 GMT
View Forum Message <> Reply to Message

On Fri, 2008-04-25 at 13:04 +0200, Pavel Machek wrote:
> Hi!
>
> > 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>
>
> There's no problem with doing this... but you should get some debate
> (with Linus?) whether using freezer for cgroups is sane. When that is

 OK, I've sent this reply directly to Linus. Hopefully this time he'll
let us know...

 One potential mitigating factor: I don't think we need the full freezer
for checkpoint/restart. Right now, because it shares code with "power
management" it's convenient to reuse the freezer. I'm hopeful that once
the freezer is no longer necessary for power management some code paths
can be simplified since I don't think checkpoint/restart requires

freezing kernel threads.

> done, there's no problem with this going in, probably through rafael's
> patch queue.

OK, if all goes well then I'll send the next round to Rafael and Cc the rest. If anyone currently on Cc doesn't care to see that feel free to let me know.

> (The first patch -- add freezer for all archs -- is probably
> reasonably to go in ASAP, through akpm or something...)
> Pavel

Well, that should only go in if the subsequent patches go in, correct? Also, since to the best of my knowledge this flag hasn't been in every arch, I'm wondering if I should Cc arch maintainers?

Cheers,
-Matt Helsley

Subject: Re: [RFC][PATCH 3/5] Container Freezer: Implement freezer cgroup subsystem
Posted by Matt Helsley on Wed, 30 Apr 2008 10:39:45 GMT
View Forum Message <> Reply to Message

On Thu, 2008-04-24 at 22:51 -0700, Paul Menage wrote:
> >+static const char *freezer_state_strs[] = {
> >+ "RUNNING\n",
> >+ "FREEZING\n" ,
> >+ "FROZEN\n"
> >+};
>
> I think it might be cleaner to not include the \n characters in this array.

Sure. Though that might produce weird output from simple_read_from_buffer() -- no newline.

I've switched this and the strcmp() code below.

> >+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);
> >+}
>
> You need to be in an RCU critical section or else hold task_lock() in
> order to dereference the cgroup returned from task_cgroup()

What are the rules of using subsystem pointers from the cgroup? Suppose
I did:

rcu_read_lock();
cgroup = task_cgroup(task, freezer_subsys_id);
freezer = cgroup_freezer(cgroup);
state = freezer->state;
rcu_read_unlock();

return (state == STATE_FROZEN);

(And guard writes to freezer->state with the freezer->lock)

?

> I'm not sure that you need to take freezer->lock here - you're just
> reading a single word.

Doesn't the safety of that assumption depend on the architecture _and_
compiler?

> >+
> >+ if (!capable(CAP_SYS_ADMIN))
> >+ return ERR_PTR(-EPERM);
> >+
>
> Why does everyone keep throwing calls to check CAP_SYS_ADMIN into
> their cgroup create callbacks? You have to be root in order to mount a
> cgroups hierarchy in the first place, and filesystem permissions will
> control who can create new cgroups.

Removed.

> >+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;
> >+}
>
> You should comment here that 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.

OK. I used your comment. :)

> >+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)
> >+{
> >+ char *buffer;
> >+ int retval = 0;
> >+ enum freezer_state goal_state;
> >+
> >+ if (nbytes >= PATH_MAX)
> >+ return -E2BIG;
> >+
> >+ /* +1 for nul-terminator */
> >+ buffer = kmalloc(nbytes + 1, GFP_KERNEL);
> >+ if (buffer == NULL)
> >+ return -ENOMEM;
>
> Given that you're copying a string whose maximum valid length is
> "FREEZING" you don't really need to use a dynamically-allocated
> buffer.

Yup. Changed to use a fixed buffer.

> But I really ought to provide a write_string() method that handles
> this kind of copying on behalf of cgroup subsystems, the way it
> already does for 64-bit ints.

Seems like a good idea for this cgroup subsystem at least.

> >+ if (strcmp(buffer, "RUNNING") == 0)
> >+ goal_state = STATE_RUNNING;
> >+ else if (strcmp(buffer, "FROZEN") == 0)
> >+ goal_state = STATE_FROZEN;
>
> Would it make sense to compare against the strings you already have in
> the array earlier in the file?

Done.

Cheers,
 -Matt Helsley

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

---

Subject: Re: [RFC][PATCH 5/5] Add a Signal Control Group Subsystem
Posted by Matt Helsley on Wed, 30 Apr 2008 18:44:57 GMT
View Forum Message <> Reply to Message

On Fri, 2008-04-25 at 13:41 +0200, Cedric Le Goater wrote:
> Matt Helsley wrote:
> > Add a signal control group subsystem that allows us to send signals to all tasks
> > in the control group by writing the desired signal(7) number to the kill file.
> >
> > NOTE: We don't really need per-cgroup state, but control groups doesn't support
> > stateless subsystems yet.
> >
> > Signed-off-by: Matt Helsley <matthltc@us.ibm.com>
> > ---
> > include/linux/cgroup_signal.h | 28 +++++++++
> > include/linux/cgroup_subsys.h | 6 +
> > init/Kconfig | 6 +
> > kernel/Makefile | 1
> > kernel/cgroup_signal.c | 129 ++
> > 5 files changed, 170 insertions(+)
> >
> > Index: linux-2.6.25-mm1/include/linux/cgroup_signal.h
> > ===
> > --- /dev/null
> > +++ linux-2.6.25-mm1/include/linux/cgroup_signal.h
> > @@ -0,0 +1,28 @@
> > +#ifndef _LINUX_CGROUP_SIGNAL_H
> > +#define _LINUX_CGROUP_SIGNAL_H
> > +/*

> > + * cgroup_signal.h - control group freezer subsystem interface
>
> s/freezer/signal/
>
> > + *
> > + * Copyright IBM Corp. 2007
> > + *
> > + * Author : Cedric Le Goater <clg@fr.ibm.com>
> > + * Author : Matt Helsley <matthltc@us.ibm.com>
> > + */
> > +
> > +#include <linux/cgroup.h>
> > +
> > +#ifdef CONFIG_CGROUP_SIGNAL
> > +
> > +struct stateless {
> > + struct cgroup_subsys_state css;
> > +};
>
> I'm not sure this is correct to say so. Imagine you want to send
> a SIGKILL to a cgroup, you would expect all tasks to die and the
> cgroup to become empty. right ?
>
> but if a task is doing clone() while it's being killed by this cgroup
> signal subsystem, we can miss the child. This is because there's a
> small window in copy_process() where the child is in the cgroup and
> not visible yet.
>
>   copy_process()
> cgroup_fork()
>   do stuff
> cgroup_fork_callbacks()
>
> cgroup_post_fork()
>   put new task in the list.
>
> (I didn't dig too much the code, though. So I might be missing
>   something)
>
> So if we want to send the signal to all tasks in the cgroup, we need
> to track the new tasks with a fork callback, just like the freezer :
>
> static void signal_fork(struct cgroup_subsys *ss, struct task_struct *task)
> {
>
> }
>
> and, of course, we need to keep somewhere the signal number we need to

> send.
>
>
> All this depends on how we want the cgroup signal subsystem to behave.
> It could be brainless of course, but it seems to me that the biggest
> benefit of such a subsystem is to use the cgroup capability to track
> new tasks coming in.
>
> Cheers,
>
> C.

Assuming we did this, isn't it still possible to send SIGSTOP to every
task in the cgroup yet still appear to have not stopped every task in
the cgroup:

```
 Task A      Task B
echo 19 > signal.send
 record signal
 return -EBUSY from can_attach
 send signals to all the tasks
 return 0 from write syscall
      echo newpid > tasks
cat tasks
<Uh oh, not all tasks are stopped...>
```

Cheers,
 -Matt Helsley

Subject: Re: [RFC][PATCH 3/5] Container Freezer: Implement freezer cgroup subsystem
Posted by Matt Helsley on Wed, 30 Apr 2008 21:28:25 GMT
View Forum Message <> Reply to Message

On Wed, 2008-04-23 at 23:47 -0700, Matt Helsley wrote:

<snip>

> +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)
> +{
> + char *buffer;
> + int retval = 0;
> + enum freezer_state goal_state;
> +
> + 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 */
> + strstrip(buffer);
> + if (strcmp(buffer, "RUNNING") == 0)
> +  goal_state = STATE_RUNNING;
> + else if (strcmp(buffer, "FROZEN") == 0)
> +  goal_state = STATE_FROZEN;
> + else {
> +  retval = -EIO;
> +  goto free_buffer;
> + }
> +
> + cgroup_lock();
> +
> + if (cgroup_is_removed(cgroup)) {
> +  retval = -ENODEV;
> +  goto unlock;
> + }
```

I think this was copy/paste'd from cgroup_common_file_write() which modifies the cgroup hierarchy. However this function does not modify the cgroup hierarchy and we're not getting the cgroup from the task. So I don't think cgroup_lock()/unlock() are needed here. Paul, do you agree?

```
> + retval = freezer_freeze(cgroup, goal_state);
> + if (retval == 0)
> +  retval = nbytes;
> +unlock:
> + cgroup_unlock();
> +free_buffer:
> + kfree(buffer);
```

> + return retval;
> +}

Cheers,
 -Matt Helsley

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

---

Subject: Re: [RFC][PATCH 3/5] Container Freezer: Implement freezer cgroup subsystem
Posted by Matt Helsley on Wed, 30 Apr 2008 22:30:49 GMT
View Forum Message <> Reply to Message

On Wed, 2008-04-30 at 14:28 -0700, Matt Helsley wrote:
> On Wed, 2008-04-23 at 23:47 -0700, Matt Helsley wrote:
>
> <snip>

> > + cgroup_lock();
> > +
> > + if (cgroup_is_removed(cgroup)) {
> > + retval = -ENODEV;
> > + goto unlock;
> > + }
>
> I think this was copy/paste'd from cgroup_common_file_write() which
> modifies the cgroup hierarchy. However this function does not modify the
> cgroup hierarchy and we're not getting the cgroup from the task. So I
> don't think cgroup_lock()/unlock() are needed here. Paul, do you agree?

Oops! I didn't track the call chain correctly. Sorry for the noise...

Cheers,
 -Matt Helsley

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

---