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_struct(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);
```

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