Subject: [RFC][PATCH 2/4] container freezer: make refrigerator() available to all archs
Posted by Cedric Le Goater on Wed, 20 Jun 2007 15:08:01 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

Now that the TIF_FREEZE flag is available in all architectures,
extract the refrigerator from kernel/power/process.c and
make it available to all. The patch also exports all the
helper routines in freezer.h

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

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
include/linux/freezer.h | 17 ++++----------
kernel/Makefile | 2 -
kernel/freezer.c | 58 ++
kernel/power/process.c | 46 ----------------------------------
4 files changed, 64 insertions(+), 59 deletions(-)

Index: 2.6.22-rc4-mm2/include/linux/freezer.h
===
--- 2.6.22-rc4-mm2.orig/include/linux/freezer.h
+++ 2.6.22-rc4-mm2/include/linux/freezer.h
@@ -5,7 +5,6 @@

#include <linux/sched.h>

-#ifdef CONFIG_PM
/*
* Check if a process has been frozen
*/
@@ -62,8 +61,6 @@ static inline int thaw_process(struct ta
}

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

static inline int try_to_freeze(void)
{
@@ -74,6 +71,11 @@ static inline int try_to_freeze(void)
return 0;
}

```
+#ifdef CONFIG_PM
+
+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
@@ -127,18 +129,9 @@ static inline void set_freezable(void)
 }

 #else
-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; }
Index: 2.6.22-rc4-mm2/kernel/Makefile
===================================================================
--- 2.6.22-rc4-mm2.orig/kernel/Makefile
+++ 2.6.22-rc4-mm2/kernel/Makefile
@@ -9,7 +9,7 @@ obj-y     = sched.o fork.o exec_domain.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 latency.o nsproxy.o srcu.o die_notifier.o \
-     utsname.o
+     utsname.o freezer.o

 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y += time/
Index: 2.6.22-rc4-mm2/kernel/freezer.c
===================================================================
--- /dev/null
+++ 2.6.22-rc4-mm2/kernel/freezer.c
@@ -0,0 +1,58 @@
+/*
+ * 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);
Index: 2.6.22-rc4-mm2/kernel/power/process.c
===================================================================
--- 2.6.22-rc4-mm2.orig/kernel/power/process.c
+++ 2.6.22-rc4-mm2/kernel/power/process.c
@@ -31,50 +31,6 @@ static inline int freezeable(struct task
  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 freeze_task(struct task_struct *p)
 {
  unsigned long flags;
@@ -226,5 +182,3 @@ void thaw_processes(void)
  schedule();
  printk("done.\n");
 }
-
-EXPORT_SYMBOL(refrigerator);

--
```