Subject: [PATCH, v6 0/3] Introduce timer slack controller
Posted by Kirill A. Shutsemov on Mon, 14 Feb 2011 13:06:24 GMT
View Forum Message <> Reply to Message

From: Kirill A. Shutemov <kirill@shutemov.name>

Changelog:

v6:
 - add documentation
 - use notifier_call_chain() instead of check hook
 - fix validate_change()
 - cleanup
v5:
 - -EBUSY on writing to timer_slack.min_slack_ns/max_slack_ns if a child has
   wider min-max range
v4:
 - hierarchy support
 - drop dummy_timer_slack_check()
 - workaround lockdep false (?) positive
 - allow 0 as timer slack value
v3:
 - rework interface
 - s/EXPORT_SYMBOL/EXPORT_SYMBOL_GPL/
v2:
 - fixed with CONFIG_CGROUP_TIMER_SLACK=y
v1:
 - initial revision

Kirill A. Shutemov (3):
  cgroups: export cgroup_iter_{start,next,end}
  Implement timer slack notifier chain
  cgroups: introduce timer slack controller

 Documentation/cgroups/timer_slack.txt | 93 ++++++++++
 include/linux/cgroup_subsys.h | 6 +
 include/linux/sched.h | 5 +
 init/Kconfig | 10 ++
 kernel/Makefile | 3 +-
 kernel/cgroup.c | 3 +
 kernel/cgroup_timer_slack.c | 285 ++++++++++++++++++++++++++++++++++
 kernel/sys.c | 9 +-
 kernel/timer_slack.c | 57 +++++++
 9 files changed, 463 insertions(+), 8 deletions(-)
 create mode 100644 Documentation/cgroups/timer_slack.txt
 create mode 100644 kernel/cgroup_timer_slack.c
 create mode 100644 kernel/timer_slack.c

--
1.7.4

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

---

Subject: [PATCH, v6 2/3] Implement timer slack notifier chain
Posted by Kirill A. Shutsemov on Mon, 14 Feb 2011 13:06:26 GMT
View Forum Message <> Reply to Message

From: Kirill A. Shutemov <kirill@shutemov.name>

Process can change its timer slack using prctl(). Timer slack notifier
call chain allows to react on such change or forbid it.

Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
---
include/linux/sched.h | 5 ++++
kernel/Makefile | 2 +-
kernel/sys.c | 9 +------
kernel/timer_slack.c | 57 ++
4 files changed, 65 insertions(+), 8 deletions(-)
create mode 100644 kernel/timer_slack.c

diff --git a/include/linux/sched.h b/include/linux/sched.h
index d747f94..f1a4ec2 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -2620,6 +2620,11 @@ static inline unsigned long rlimit_max(unsigned int limit)
return task_rlimit_max(current, limit);
}

+extern int register_timer_slack_notifier(struct notifier_block *nb);
+extern void unregister_timer_slack_notifier(struct notifier_block *nb);
+long prctl_get_timer_slack(void);
+extern long prctl_set_timer_slack(long timer_slack_ns);
+
#endif /* __KERNEL__ */

#endif
diff --git a/kernel/Makefile b/kernel/Makefile
index 353d3fe..aa43e63 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile

```
@@ -10,7 +10,7 @@ obj-y     = sched.o fork.o exec_domain.o panic.o printk.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 sched_clock.o cred.o \
-      async.o range.o jump_label.o
+      async.o range.o jump_label.o timer_slack.o
  obj-y += groups.o

  ifdef CONFIG_FUNCTION_TRACER
diff --git a/kernel/sys.c b/kernel/sys.c
index 18da702..a33e4b8 100644
--- a/kernel/sys.c
+++ b/kernel/sys.c
@@ -1691,15 +1691,10 @@ SYSCALL_DEFINE5(prctl, int, option, unsigned long, arg2,
unsigned long, arg3,
     error = perf_event_task_enable();
     break;
    case PR_GET_TIMERSLACK:
-    error = current->timer_slack_ns;
+    error = prctl_get_timer_slack();
     break;
    case PR_SET_TIMERSLACK:
-    if (arg2 <= 0)
-     current->timer_slack_ns =
-      current->default_timer_slack_ns;
-    else
-     current->timer_slack_ns = arg2;
-    error = 0;
+    error = prctl_set_timer_slack(arg2);
     break;
    case PR_MCE_KILL:
     if (arg4 | arg5)
diff --git a/kernel/timer_slack.c b/kernel/timer_slack.c
new file mode 100644
index 0000000..546ed78
--- /dev/null
+++ b/kernel/timer_slack.c
@@ -0,0 +1,57 @@
+/*
+ * kernel/timer_slack.c
+ *
+ * Copyright Nokia Corparation, 2011
+ * Author: Kirill A. Shutemov
+ *
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
```

```
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ */
+
+#include <linux/notifier.h>
+#include <linux/module.h>
+#include <linux/sched.h>
+
+static BLOCKING_NOTIFIER_HEAD(timer_slack_notify_list);
+
+int register_timer_slack_notifier(struct notifier_block *nb)
+{
+ return  blocking_notifier_chain_register(&timer_slack_notify_lis t, nb);
+}
+EXPORT_SYMBOL_GPL(register_timer_slack_notifier);
+
+void unregister_timer_slack_notifier(struct notifier_block *nb)
+{
+  blocking_notifier_chain_unregister(&timer_slack_notify_l ist, nb);
+}
+EXPORT_SYMBOL_GPL(unregister_timer_slack_notifier);
+
+long prctl_get_timer_slack(void)
+{
+ return current->timer_slack_ns;
+}
+
+long prctl_set_timer_slack(long timer_slack_ns)
+{
+ int err;
+
+ /* Reset timer slack to default value */
+ if (timer_slack_ns <= 0) {
+  current->timer_slack_ns = current->default_timer_slack_ns;
+  return 0;
+ }
+
+ err = blocking_notifier_call_chain(&timer_slack_notify_list,
+   timer_slack_ns, NULL);
+ if (err == NOTIFY_DONE)
+  current->timer_slack_ns = timer_slack_ns;
+
+ return notifier_to_errno(err);
+}
--
```

1.7.4

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

---

Subject: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Kirill A. Shutsemov on Mon, 14 Feb 2011 13:06:27 GMT
View Forum Message <> Reply to Message

From: Kirill A. Shutemov <kirill@shutemov.name>

Every task_struct has timer_slack_ns value. This value uses to round up
poll() and select() timeout values. This feature can be useful in
mobile environment where combined wakeups are desired.

cgroup subsys "timer_slack" implement timer slack controller. It
provides a way to group tasks by timer slack value and manage the
value of group's tasks.

Idea-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>

Documentation/cgroups/timer_slack.txt | 93 ++++++++++++
include/linux/cgroup_subsys.h | 6 +
init/Kconfig | 10 ++
kernel/Makefile | 1 +
kernel/cgroup_timer_slack.c | 285 ++++++++++++++++++++++++++++++++++++++
5 files changed, 395 insertions(+), 0 deletions(-)
create mode 100644 Documentation/cgroups/timer_slack.txt
create mode 100644 kernel/cgroup_timer_slack.c

diff --git a/Documentation/cgroups/timer_slack.txt b/Documentation/cgroups/timer_slack.txt
new file mode 100644
index 0000000..e7ec2f3
--- /dev/null
+++ b/Documentation/cgroups/timer_slack.txt
@@ -0,0 +1,93 @@
+Timer Slack Controller
+======================
+
+Overview
+--------
+
+Every task_struct has timer_slack_ns value. This value uses to round up
+poll() and select() timeout values. This feature can be useful in

+mobile environment where combined wakeups are desired.
+
+cgroup subsys "timer_slack" implement timer slack controller. It
+provides a way to group tasks by timer slack value and manage the
+value of group's tasks.
+
+
+User interface
+--------------
+
+To get timer slack controller functionality you need to enable it in
+kernel configuration:
+
+CONFIG_CGROUP_TIMER_SLACK=y
+
+or if you want to compile it as module:
+
+CONFIG_CGROUP_TIMER_SLACK=m
+
+The controller provides three files in cgroup directory:
+
+# mount -t cgroup -o timer_slack none /sys/fs/cgroup
+# ls /sys/fs/cgroup/timer_slack.*
+/sys/fs/cgroup/timer_slack.max_slack_ns
+/sys/fs/cgroup/timer_slack.min_slack_ns
+/sys/fs/cgroup/timer_slack.set_slack_ns
+
+timer_slack.min_slack_ns and timer_slack.set_slack_ns specify allowed
+range of timer slack for tasks in cgroup. By default it unlimited:
+
+# cat /sys/fs/cgroup/timer_slack.min_slack_ns
+0
+# cat /sys/fs/cgroup/timer_slack.max_slack_ns
+4294967295
+
+You can specify limits you want:
+
+# echo 50000 > /sys/fs/cgroup/timer_slack.min_slack_ns
+# echo 1000000 > /sys/fs/cgroup/timer_slack.max_slack_ns
+# cat /sys/fs/cgroup/timer_slack.{min,max}_slack_ns
+50000
+1000000
+
+Timer slack value of all tasks of the cgroup will be adjusted to fit
+min-max range.
+
+If a task will try to call prctl() to change timer slack value out of
+the range it get -EPERM.

```
+
+You can change timer slack value of all tasks of the cgroup at once:
+
+# echo 70000 > /sys/fs/cgroup/timer_slack.set_slack_ns
+
+Timer slack controller supports hierarchical groups. The only rule:
+parent's limit range should be wider or equal to child's. Sibling
+cgroups can have overlapping min-max range.
+
+  (root: 50000 - 1000000)
+   /       \
+ (a: 50000 - 50000) (b: 500000 - 1000000)
+     /     \
+   (c: 500000 - 900000) (d: 700000 - 800000)
+
+# mkdir /sys/fs/cgroup/a
+# echo 50000 > /sys/fs/cgroup/a/timer_slack.max_slack_ns
+# cat /sys/fs/cgroup/a/timer_slack.{min,max}_slack_ns
+50000
+50000
+# mkdir /sys/fs/cgroup/b
+# echo 500000 > /sys/fs/cgroup/b/timer_slack.min_slack_ns
+# cat /sys/fs/cgroup/b/timer_slack.{min,max}_slack_ns
+500000
+1000000
+# mkdir /sys/fs/cgroup/b/c
+# echo 900000 > /sys/fs/cgroup/b/c/timer_slack.max_slack_ns
+# cat /sys/fs/cgroup/b/c/timer_slack.{min,max}_slack_ns
+500000
+900000
+# mkdir /sys/fs/cgroup/b/d
+# echo 700000 > /sys/fs/cgroup/b/d/timer_slack.min_slack_ns
+# echo 800000 > /sys/fs/cgroup/b/d/timer_slack.max_slack_ns
+# cat /sys/fs/cgroup/b/d/timer_slack.{min,max}_slack_ns
+700000
+800000
+
diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
index ccefff0..e399228 100644
--- a/include/linux/cgroup_subsys.h
+++ b/include/linux/cgroup_subsys.h
@@ -66,3 +66,9 @@ SUBSYS(blkio)
 #endif

 /* */
+
+#ifdef CONFIG_CGROUP_TIMER_SLACK
+SUBSYS(timer_slack)
```

```
+#endif
+
+/* */
diff --git a/init/Kconfig b/init/Kconfig
index be788c0..bb54ae9 100644
--- a/init/Kconfig
+++ b/init/Kconfig
@@ -596,6 +596,16 @@ config CGROUP_FREEZER
    Provides a way to freeze and unfreeze all tasks in a
    cgroup.

+config CGROUP_TIMER_SLACK
+ tristate "Timer slack cgroup controller"
+ help
+   Provides a way of tasks grouping by timer slack value.
+   Every timer slack cgroup has min and max slack value. Task's
+   timer slack value will be adjusted to fit min-max range once
+   the task attached to the cgroup.
+   It's useful in mobile devices where certain background apps
+   are attached to a cgroup and combined wakeups are desired.
+
 config CGROUP_DEVICE
  bool "Device controller for cgroups"
  help
diff --git a/kernel/Makefile b/kernel/Makefile
index aa43e63..0e660c5 100644
--- a/kernel/Makefile
+++ b/kernel/Makefile
@@ -61,6 +61,7 @@ obj-$(CONFIG_BACKTRACE_SELF_TEST) += backtracetest.o
 obj-$(CONFIG_COMPAT) += compat.o
 obj-$(CONFIG_CGROUPS) += cgroup.o
 obj-$(CONFIG_CGROUP_FREEZER) += cgroup_freezer.o
+obj-$(CONFIG_CGROUP_TIMER_SLACK) += cgroup_timer_slack.o
 obj-$(CONFIG_CPUSETS) += cpuset.o
 obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
 obj-$(CONFIG_UTS_NS) += utsname.o
diff --git a/kernel/cgroup_timer_slack.c b/kernel/cgroup_timer_slack.c
new file mode 100644
index 0000000..ec63700
--- /dev/null
+++ b/kernel/cgroup_timer_slack.c
@@ -0,0 +1,285 @@
+/*
+ * cgroup_timer_slack.c - control group timer slack subsystem
+ *
+ * Copyright Nokia Corparation, 2011
+ * Author: Kirill A. Shutemov
+ *
```

```
+ * This program is free software; you can redistribute it and/or modify
+ * it under the terms of the GNU General Public License as published by
+ * the Free Software Foundation; either version 2 of the License, or
+ * (at your option) any later version.
+ *
+ * This program is distributed in the hope that it will be useful,
+ * but WITHOUT ANY WARRANTY; without even the implied warranty of
+ * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
+ * GNU General Public License for more details.
+ */
+#include <linux/cgroup.h>
+#include <linux/init_task.h>
+#include <linux/module.h>
+#include <linux/slab.h>
+#include <linux/rcupdate.h>
+
+struct cgroup_subsys timer_slack_subsys;
+struct timer_slack_cgroup {
+ struct cgroup_subsys_state css;
+ unsigned long min_slack_ns;
+ unsigned long max_slack_ns;
+};
+
+enum {
+ TIMER_SLACK_MIN,
+ TIMER_SLACK_MAX,
+};
+
+static struct timer_slack_cgroup *cgroup_to_tslack_cgroup(struct cgroup *cgroup)
+{
+ struct cgroup_subsys_state *css;
+
+ css = cgroup_subsys_state(cgroup, timer_slack_subsys.subsys_id);
+ return container_of(css, struct timer_slack_cgroup, css);
+}
+
+static int is_timer_slack_allowed(struct timer_slack_cgroup *tslack_cgroup,
+  unsigned long slack_ns)
+{
+ if (slack_ns < tslack_cgroup->min_slack_ns ||
+    slack_ns > tslack_cgroup->max_slack_ns)
+  return false;
+ return true;
+}
+
+static int cgroup_timer_slack_check(struct notifier_block *nb,
+  unsigned long slack_ns, void *data)
+{
```

```
+ struct cgroup_subsys_state *css;
+ struct timer_slack_cgroup *tslack_cgroup;
+
+ /* XXX: lockdep false positive? */
+ rcu_read_lock();
+ css = task_subsys_state(current, timer_slack_subsys.subsys_id);
+ tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
+ rcu_read_unlock();
+
+ if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
+  return notifier_from_errno(-EPERM);
+ return NOTIFY_OK;
+}
+
+static struct notifier_block cgroup_timer_slack_nb = {
+ .notifier_call = cgroup_timer_slack_check,
+};
+
+static struct cgroup_subsys_state *
+tslack_cgroup_create(struct cgroup_subsys *subsys, struct cgroup *cgroup)
+{
+ struct timer_slack_cgroup *tslack_cgroup;
+
+ tslack_cgroup = kmalloc(sizeof(*tslack_cgroup), GFP_KERNEL);
+ if (!tslack_cgroup)
+  return ERR_PTR(-ENOMEM);
+
+ if (cgroup->parent) {
+  struct timer_slack_cgroup *parent;
+  parent = cgroup_to_tslack_cgroup(cgroup->parent);
+  tslack_cgroup->min_slack_ns = parent->min_slack_ns;
+  tslack_cgroup->max_slack_ns = parent->max_slack_ns;
+ } else {
+  tslack_cgroup->min_slack_ns = 0UL;
+  tslack_cgroup->max_slack_ns = ULONG_MAX;
+ }
+
+ return &tslack_cgroup->css;
+}
+
+static void tslack_cgroup_destroy(struct cgroup_subsys *subsys,
+  struct cgroup *cgroup)
+{
+ kfree(cgroup_to_tslack_cgroup(cgroup));
+}
+
+/*
+ * Adjust ->timer_slack_ns and ->default_max_slack_ns of the task to fit
```

```
+ * limits of the cgroup.
+ */
+static void tslack_adjust_task(struct timer_slack_cgroup *tslack_cgroup,
+  struct task_struct *tsk)
+{
+ if (tslack_cgroup->min_slack_ns > tsk->timer_slack_ns)
+  tsk->timer_slack_ns = tslack_cgroup->min_slack_ns;
+ else if (tslack_cgroup->max_slack_ns < tsk->timer_slack_ns)
+  tsk->timer_slack_ns = tslack_cgroup->max_slack_ns;
+
+ if (tslack_cgroup->min_slack_ns > tsk->default_timer_slack_ns)
+  tsk->default_timer_slack_ns = tslack_cgroup->min_slack_ns;
+ else if (tslack_cgroup->max_slack_ns < tsk->default_timer_slack_ns)
+  tsk->default_timer_slack_ns = tslack_cgroup->max_slack_ns;
+}
+
+static void tslack_cgroup_attach(struct cgroup_subsys *subsys,
+  struct cgroup *cgroup, struct cgroup *prev,
+  struct task_struct *tsk, bool threadgroup)
+{
+ tslack_adjust_task(cgroup_to_tslack_cgroup(cgroup), tsk);
+}
+
+static int tslack_write_set_slack_ns(struct cgroup *cgroup, struct cftype *cft,
+  u64 val)
+{
+ struct timer_slack_cgroup *tslack_cgroup;
+ struct cgroup_iter it;
+ struct task_struct *task;
+
+ tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
+ if (!is_timer_slack_allowed(cgroup_to_tslack_cgroup(cgroup), val))
+  return -EPERM;
+
+ /* Change timer slack value for all tasks in the cgroup */
+ cgroup_iter_start(cgroup, &it);
+ while ((task = cgroup_iter_next(cgroup, &it)))
+  task->timer_slack_ns = val;
+ cgroup_iter_end(cgroup, &it);
+
+ return 0;
+}
+
+static u64 tslack_read_range(struct cgroup *cgroup, struct cftype *cft)
+{
+ struct timer_slack_cgroup *tslack_cgroup;
+
+ tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
```

```
+ switch (cft->private) {
+ case TIMER_SLACK_MIN:
+  return tslack_cgroup->min_slack_ns;
+ case TIMER_SLACK_MAX:
+  return tslack_cgroup->max_slack_ns;
+ default:
+  BUG();
+ }
+}
+
+static int validate_change(struct cgroup *cgroup, u64 val, int type)
+{
+ struct timer_slack_cgroup *tslack_cgroup, *child;
+ struct cgroup *cur;
+
+ BUG_ON(type != TIMER_SLACK_MIN && type != TIMER_SLACK_MAX);
+
+ if (val > ULONG_MAX)
+  return -EINVAL;
+
+ if (cgroup->parent) {
+  struct timer_slack_cgroup *parent;
+  parent = cgroup_to_tslack_cgroup(cgroup->parent);
+  if (!is_timer_slack_allowed(parent, val))
+   return -EPERM;
+ }
+
+ tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
+ if (type == TIMER_SLACK_MIN && val > tslack_cgroup->max_slack_ns)
+  return -EINVAL;
+ if (type == TIMER_SLACK_MAX && val < tslack_cgroup->min_slack_ns)
+  return -EINVAL;
+
+ list_for_each_entry(cur, &cgroup->children, sibling) {
+  child = cgroup_to_tslack_cgroup(cur);
+  if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
+   return -EBUSY;
+  if (type == TIMER_SLACK_MAX && val < child->max_slack_ns)
+   return -EBUSY;
+ }
+
+ return 0;
+}
+
+static int tslack_write_range(struct cgroup *cgroup, struct cftype *cft,
+  u64 val)
+{
+ struct timer_slack_cgroup *tslack_cgroup;
```

```
+ struct cgroup_iter it;
+ struct task_struct *task;
+ int err;
+
+ err = validate_change(cgroup, val, cft->private);
+ if (err)
+  return err;
+
+ tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
+ if (cft->private == TIMER_SLACK_MIN)
+  tslack_cgroup->min_slack_ns = val;
+ else
+  tslack_cgroup->max_slack_ns = val;
+
+ /*
+  * Adjust timer slack value for all tasks in the cgroup to fit
+  * min-max range.
+  */
+ cgroup_iter_start(cgroup, &it);
+ while ((task = cgroup_iter_next(cgroup, &it)))
+  tslack_adjust_task(tslack_cgroup, task);
+ cgroup_iter_end(cgroup, &it);
+
+ return 0;
+}
+
+static struct cftype files[] = {
+ {
+  .name = "set_slack_ns",
+  .write_u64 = tslack_write_set_slack_ns,
+ },
+ {
+  .name = "min_slack_ns",
+  .private = TIMER_SLACK_MIN,
+  .read_u64 = tslack_read_range,
+  .write_u64 = tslack_write_range,
+ },
+ {
+  .name = "max_slack_ns",
+  .private = TIMER_SLACK_MAX,
+  .read_u64 = tslack_read_range,
+  .write_u64 = tslack_write_range,
+ },
+};
+
+static int tslack_cgroup_populate(struct cgroup_subsys *subsys,
+  struct cgroup *cgroup)
+{
```

```
+ return cgroup_add_files(cgroup, subsys, files, ARRAY_SIZE(files));
+}
+
+struct cgroup_subsys timer_slack_subsys = {
+ .name = "timer_slack",
+ .module = THIS_MODULE,
+#ifndef CONFIG_CGROUP_TIMER_SLACK_MODULE
+ .subsys_id = timer_slack_subsys_id,
+#endif
+ .create = tslack_cgroup_create,
+ .destroy = tslack_cgroup_destroy,
+ .attach = tslack_cgroup_attach,
+ .populate = tslack_cgroup_populate,
+};
+
+static int __init init_cgroup_timer_slack(void)
+{
+ int err;
+
+ err = register_timer_slack_notifier(&cgroup_timer_slack_nb);
+ if (err)
+  return err;
+
+ err = cgroup_load_subsys(&timer_slack_subsys);
+ if (err)
+   unregister_timer_slack_notifier(&cgroup_timer_slack_nb);
+
+ return err;
+}
+
+static void __exit exit_cgroup_timer_slack(void)
+{
+  unregister_timer_slack_notifier(&cgroup_timer_slack_nb);
+ cgroup_unload_subsys(&timer_slack_subsys);
+}
+
+module_init(init_cgroup_timer_slack);
+module_exit(exit_cgroup_timer_slack);
+MODULE_LICENSE("GPL");
--
1.7.4
```

Subject: Re: [PATCH, v6 0/3] Introduce timer slack controller
Posted by Thomas Gleixner on Mon, 14 Feb 2011 13:26:42 GMT
View Forum Message <> Reply to Message

On Mon, 14 Feb 2011, Kirill A. Shutsemov wrote:

This 0/0 explains nothing nada. What is the whole muck about ?

Thanks,

tglx

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

Subject: Re: [PATCH, v6 2/3] Implement timer slack notifier chain
Posted by Thomas Gleixner on Mon, 14 Feb 2011 13:32:23 GMT
View Forum Message <> Reply to Message

On Mon, 14 Feb 2011, Kirill A. Shutsemov wrote:

> From: Kirill A. Shutemov <kirill@shutemov.name>
>
> Process can change its timer slack using prctl(). Timer slack notifier
> call chain allows to react on such change or forbid it.

So we add a notifier call chain and more exports to allow what ?

> --- a/kernel/sys.c
> +++ b/kernel/sys.c
> @@ -1691,15 +1691,10 @@ SYSCALL_DEFINE5(prctl, int, option, unsigned long, arg2,
unsigned long, arg3,
> error = perf_event_task_enable();
> break;
> case PR_GET_TIMERSLACK:
> - error = current->timer_slack_ns;
> + error = prctl_get_timer_slack();

What's the point of replacing current->timer_slack_ns with a
function which does exactly the same ?

> +long prctl_set_timer_slack(long timer_slack_ns)
> +{
> + int err;
> +
> + /* Reset timer slack to default value */

> + if (timer_slack_ns <= 0) {
> + current->timer_slack_ns = current->default_timer_slack_ns;
> + return 0;

That does not make any sense at all. Why is setting
default_timer_slack_ns not subject to validation ? Why is it
treaded seperately ?

> + }
> +
> + err = blocking_notifier_call_chain(&timer_slack_notify_list,
> + timer_slack_ns, NULL);
> + if (err == NOTIFY_DONE)
> + current->timer_slack_ns = timer_slack_ns;
> +
> + return notifier_to_errno(err);

Thanks,

tglx

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

---

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Matt Helsley on Mon, 14 Feb 2011 13:59:26 GMT
View Forum Message <> Reply to Message

On Mon, Feb 14, 2011 at 03:06:27PM +0200, Kirill A. Shutsemov wrote:
> From: Kirill A. Shutemov <kirill@shutemov.name>
>
> Every task_struct has timer_slack_ns value. This value uses to round up
> poll() and select() timeout values. This feature can be useful in
> mobile environment where combined wakeups are desired.
>
> cgroup subsys "timer_slack" implement timer slack controller. It
> provides a way to group tasks by timer slack value and manage the
> value of group's tasks.
>
> Idea-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
> Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
> ---
> Documentation/cgroups/timer_slack.txt | 93 ++++++++++++
> include/linux/cgroup_subsys.h | 6 +
> init/Kconfig | 10 ++
> kernel/Makefile | 1 +

> kernel/cgroup_timer_slack.c | 285 ++++++++++++++++++++++++++++++++++++
> 5 files changed, 395 insertions(+), 0 deletions(-)
> create mode 100644 Documentation/cgroups/timer_slack.txt
> create mode 100644 kernel/cgroup_timer_slack.c
>
> diff --git a/Documentation/cgroups/timer_slack.txt b/Documentation/cgroups/timer_slack.txt
> new file mode 100644
> index 0000000..e7ec2f3
> --- /dev/null
> +++ b/Documentation/cgroups/timer_slack.txt
> @@ -0,0 +1,93 @@
> +Timer Slack Controller
> +======================
> +
> +Overview
> +--------
> +
> +Every task_struct has timer_slack_ns value. This value uses to round up
> +poll() and select() timeout values. This feature can be useful in
> +mobile environment where combined wakeups are desired.
> +
> +cgroup subsys "timer_slack" implement timer slack controller. It
> +provides a way to group tasks by timer slack value and manage the
> +value of group's tasks.
> +
> +
> +User interface
> +--------------
> +
> +To get timer slack controller functionality you need to enable it in
> +kernel configuration:
> +
> +CONFIG_CGROUP_TIMER_SLACK=y
> +
> +or if you want to compile it as module:
> +
> +CONFIG_CGROUP_TIMER_SLACK=m
> +
> +The controller provides three files in cgroup directory:
> +
> +# mount -t cgroup -o timer_slack none /sys/fs/cgroup
> +# ls /sys/fs/cgroup/timer_slack.*
> +/sys/fs/cgroup/timer_slack.max_slack_ns
> +/sys/fs/cgroup/timer_slack.min_slack_ns
> +/sys/fs/cgroup/timer_slack.set_slack_ns
> +
> +timer_slack.min_slack_ns and timer_slack.set_slack_ns specify allowed
> +range of timer slack for tasks in cgroup. By default it unlimited:

> +
> +# cat /sys/fs/cgroup/timer_slack.min_slack_ns
> +0
> +# cat /sys/fs/cgroup/timer_slack.max_slack_ns
> +4294967295
> +
> +You can specify limits you want:
> +
> +# echo 50000 > /sys/fs/cgroup/timer_slack.min_slack_ns
> +# echo 1000000 > /sys/fs/cgroup/timer_slack.max_slack_ns
> +# cat /sys/fs/cgroup/timer_slack.{min,max}_slack_ns
> +50000
> +1000000
> +
> +Timer slack value of all tasks of the cgroup will be adjusted to fit
> +min-max range.
> +
> +If a task will try to call prctl() to change timer slack value out of
> +the range it get -EPERM.
> +
> +You can change timer slack value of all tasks of the cgroup at once:
> +
> +# echo 70000 > /sys/fs/cgroup/timer_slack.set_slack_ns
> +
> +Timer slack controller supports hierarchical groups. The only rule:
> +parent's limit range should be wider or equal to child's. Sibling
> +cgroups can have overlapping min-max range.
> +
> + (root: 50000 - 1000000)
> +   /   \
> + (a: 50000 - 50000) (b: 500000 - 1000000)
> +   /   \
> +   (c: 500000 - 900000) (d: 700000 - 800000)
> +
> +# mkdir /sys/fs/cgroup/a
> +# echo 50000 > /sys/fs/cgroup/a/timer_slack.max_slack_ns
> +# cat /sys/fs/cgroup/a/timer_slack.{min,max}_slack_ns
> +50000
> +50000
> +# mkdir /sys/fs/cgroup/b
> +# echo 500000 > /sys/fs/cgroup/b/timer_slack.min_slack_ns
> +# cat /sys/fs/cgroup/b/timer_slack.{min,max}_slack_ns
> +500000
> +1000000
> +# mkdir /sys/fs/cgroup/b/c
> +# echo 900000 > /sys/fs/cgroup/b/c/timer_slack.max_slack_ns
> +# cat /sys/fs/cgroup/b/c/timer_slack.{min,max}_slack_ns
> +500000

> +900000
> +# mkdir /sys/fs/cgroup/b/d
> +# echo 700000 > /sys/fs/cgroup/b/d/timer_slack.min_slack_ns
> +# echo 800000 > /sys/fs/cgroup/b/d/timer_slack.max_slack_ns
> +# cat /sys/fs/cgroup/b/d/timer_slack.{min,max}_slack_ns
> +700000
> +800000
> +
> diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
> index ccefff0..e399228 100644
> --- a/include/linux/cgroup_subsys.h
> +++ b/include/linux/cgroup_subsys.h
> @@ -66,3 +66,9 @@ SUBSYS(blkio)
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CGROUP_TIMER_SLACK
> +SUBSYS(timer_slack)
> +#endif
> +
> +/* */
> diff --git a/init/Kconfig b/init/Kconfig
> index be788c0..bb54ae9 100644
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -596,6 +596,16 @@ config CGROUP_FREEZER
>   Provides a way to freeze and unfreeze all tasks in a
>   cgroup.
>
> +config CGROUP_TIMER_SLACK
> + tristate "Timer slack cgroup controller"
> + help
> +   Provides a way of tasks grouping by timer slack value.
> +   Every timer slack cgroup has min and max slack value. Task's
> +   timer slack value will be adjusted to fit min-max range once
> +   the task attached to the cgroup.
> +   It's useful in mobile devices where certain background apps
> +   are attached to a cgroup and combined wakeups are desired.
> +
> config CGROUP_DEVICE
>   bool "Device controller for cgroups"
>   help
> diff --git a/kernel/Makefile b/kernel/Makefile
> index aa43e63..0e660c5 100644
> --- a/kernel/Makefile
> +++ b/kernel/Makefile
> @@ -61,6 +61,7 @@ obj-$(CONFIG_BACKTRACE_SELF_TEST) += backtracetest.o

> obj-$(CONFIG_COMPAT) += compat.o
> obj-$(CONFIG_CGROUPS) += cgroup.o
> obj-$(CONFIG_CGROUP_FREEZER) += cgroup_freezer.o
> +obj-$(CONFIG_CGROUP_TIMER_SLACK) += cgroup_timer_slack.o
> obj-$(CONFIG_CPUSETS) += cpuset.o
> obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
> obj-$(CONFIG_UTS_NS) += utsname.o
> diff --git a/kernel/cgroup_timer_slack.c b/kernel/cgroup_timer_slack.c
> new file mode 100644
> index 0000000..ec63700
> --- /dev/null
> +++ b/kernel/cgroup_timer_slack.c
> @@ -0,0 +1,285 @@
> +/*
> + * cgroup_timer_slack.c - control group timer slack subsystem
> + *
> + * Copyright Nokia Corparation, 2011
> + * Author: Kirill A. Shutemov
> + *
> + * This program is free software; you can redistribute it and/or modify
> + * it under the terms of the GNU General Public License as published by
> + * the Free Software Foundation; either version 2 of the License, or
> + * (at your option) any later version.
> + *
> + * This program is distributed in the hope that it will be useful,
> + * but WITHOUT ANY WARRANTY; without even the implied warranty of
> + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
> + * GNU General Public License for more details.
> + */
> +#include <linux/cgroup.h>
> +#include <linux/init_task.h>
> +#include <linux/module.h>
> +#include <linux/slab.h>
> +#include <linux/rcupdate.h>
> +
> +struct cgroup_subsys timer_slack_subsys;
> +struct timer_slack_cgroup {
> + struct cgroup_subsys_state css;
> + unsigned long min_slack_ns;
> + unsigned long max_slack_ns;
> +};
> +
> +enum {
> + TIMER_SLACK_MIN,
> + TIMER_SLACK_MAX,
> +};
> +
> +static struct timer_slack_cgroup *cgroup_to_tslack_cgroup(struct cgroup *cgroup)

```
> +{
> + struct cgroup_subsys_state *css;
> +
> + css = cgroup_subsys_state(cgroup, timer_slack_subsys.subsys_id);
> + return container_of(css, struct timer_slack_cgroup, css);
> +}
> +
> +static int is_timer_slack_allowed(struct timer_slack_cgroup *tslack_cgroup,
> +  unsigned long slack_ns)
> +{
> + if (slack_ns < tslack_cgroup->min_slack_ns ||
> +   slack_ns > tslack_cgroup->max_slack_ns)
> +  return false;
> + return true;
> +}
> +
> +static int cgroup_timer_slack_check(struct notifier_block *nb,
> +  unsigned long slack_ns, void *data)
> +{
> + struct cgroup_subsys_state *css;
> + struct timer_slack_cgroup *tslack_cgroup;
> +
> + /* XXX: lockdep false positive? */
> + rcu_read_lock();
> + css = task_subsys_state(current, timer_slack_subsys.subsys_id);
> + tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
> + rcu_read_unlock();
> +
> + if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
```

I think this test -- or at least accesses to the cgroup's min/max values
-- needs to be in the rcu_read_lock() else there is a race between the
notifier call from the context of the task calling prctl() and the context
of the task writing to the cgroup's (min|max)_slack_ns files.

```
> +  return notifier_from_errno(-EPERM);
> + return NOTIFY_OK;
> +}
> +
> +static struct notifier_block cgroup_timer_slack_nb = {
> + .notifier_call = cgroup_timer_slack_check,
> +};
> +
> +static struct cgroup_subsys_state *
> +tslack_cgroup_create(struct cgroup_subsys *subsys, struct cgroup *cgroup)
> +{
> + struct timer_slack_cgroup *tslack_cgroup;
> +
```

```
> + tslack_cgroup = kmalloc(sizeof(*tslack_cgroup), GFP_KERNEL);
> + if (!tslack_cgroup)
> +  return ERR_PTR(-ENOMEM);
> +
> + if (cgroup->parent) {
> +  struct timer_slack_cgroup *parent;
> +  parent = cgroup_to_tslack_cgroup(cgroup->parent);
> +  tslack_cgroup->min_slack_ns = parent->min_slack_ns;
> +  tslack_cgroup->max_slack_ns = parent->max_slack_ns;
> + } else {
> +  tslack_cgroup->min_slack_ns = 0UL;
> +  tslack_cgroup->max_slack_ns = ULONG_MAX;
> + }
> +
> + return &tslack_cgroup->css;
> +}
> +
> +static void tslack_cgroup_destroy(struct cgroup_subsys *subsys,
> +  struct cgroup *cgroup)
> +{
> + kfree(cgroup_to_tslack_cgroup(cgroup));
> +}
> +
> +/*
> + * Adjust ->timer_slack_ns and ->default_max_slack_ns of the task to fit
> + * limits of the cgroup.
> + */
> +static void tslack_adjust_task(struct timer_slack_cgroup *tslack_cgroup,
> +  struct task_struct *tsk)
> +{
> + if (tslack_cgroup->min_slack_ns > tsk->timer_slack_ns)
> +  tsk->timer_slack_ns = tslack_cgroup->min_slack_ns;
> + else if (tslack_cgroup->max_slack_ns < tsk->timer_slack_ns)
> +  tsk->timer_slack_ns = tslack_cgroup->max_slack_ns;
```

It occured to me there's a macro for this in include/linux/kernel.h:

```
tsk->timer_slack_ns = clamp(tsk->timer_slack_ns,
    tslack_cgroup->min_slack_ns,
    tslack_cgroup->max_slack_ns);
```

```
> +
> + if (tslack_cgroup->min_slack_ns > tsk->default_timer_slack_ns)
> +  tsk->default_timer_slack_ns = tslack_cgroup->min_slack_ns;
> + else if (tslack_cgroup->max_slack_ns < tsk->default_timer_slack_ns)
> +  tsk->default_timer_slack_ns = tslack_cgroup->max_slack_ns;
```

```
tsk->default_timer_slack_ns = clamp(tsk->default_timer_slack_ns,
```

```
    tslack_cgroup->min_slack_ns,
    tslack_cgroup->max_slack_ns);

> +}
> +
> +static void tslack_cgroup_attach(struct cgroup_subsys *subsys,
> +  struct cgroup *cgroup, struct cgroup *prev,
> +  struct task_struct *tsk, bool threadgroup)
> +{
> + tslack_adjust_task(cgroup_to_tslack_cgroup(cgroup), tsk);
> +}
> +
> +static int tslack_write_set_slack_ns(struct cgroup *cgroup, struct cftype *cft,
> +  u64 val)
> +{
> + struct timer_slack_cgroup *tslack_cgroup;
> + struct cgroup_iter it;
> + struct task_struct *task;
> +
> + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> + if (!is_timer_slack_allowed(cgroup_to_tslack_cgroup(cgroup), val))
> +  return -EPERM;
> +
> + /* Change timer slack value for all tasks in the cgroup */
> + cgroup_iter_start(cgroup, &it);
> + while ((task = cgroup_iter_next(cgroup, &it)))
> +  task->timer_slack_ns = val;
> + cgroup_iter_end(cgroup, &it);
> +
> + return 0;
> +}
> +
> +static u64 tslack_read_range(struct cgroup *cgroup, struct cftype *cft)
> +{
> + struct timer_slack_cgroup *tslack_cgroup;
> +
> + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> + switch (cft->private) {
> + case TIMER_SLACK_MIN:
> +  return tslack_cgroup->min_slack_ns;
> + case TIMER_SLACK_MAX:
> +  return tslack_cgroup->max_slack_ns;
> + default:
> +  BUG();
> + }
> +}
> +
> +static int validate_change(struct cgroup *cgroup, u64 val, int type)
```

```
> +{
> + struct timer_slack_cgroup *tslack_cgroup, *child;
> + struct cgroup *cur;
> +
> + BUG_ON(type != TIMER_SLACK_MIN && type != TIMER_SLACK_MAX);
> +
> + if (val > ULONG_MAX)
> +  return -EINVAL;
> +
> + if (cgroup->parent) {
> +  struct timer_slack_cgroup *parent;
> +  parent = cgroup_to_tslack_cgroup(cgroup->parent);
> +  if (!is_timer_slack_allowed(parent, val))
> +   return -EPERM;
> + }
> +
> + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> + if (type == TIMER_SLACK_MIN && val > tslack_cgroup->max_slack_ns)
> +  return -EINVAL;
> + if (type == TIMER_SLACK_MAX && val < tslack_cgroup->min_slack_ns)
> +  return -EINVAL;
> +
> + list_for_each_entry(cur, &cgroup->children, sibling) {
> +  child = cgroup_to_tslack_cgroup(cur);
> +  if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
> +   return -EBUSY;
> +  if (type == TIMER_SLACK_MAX && val < child->max_slack_ns)
> +   return -EBUSY;
> + }
```

This doesn't look right. Child cgroups should not constrain their parents. Instead you should allow the change and propagate the constraint to the children.

One thing that might make reviewing such changes to these patches easier would be to split out the arbitrary-depth hierarchy support into a follow-on patch. You can do it like blkio does in the _create() function:

```
        /* Currently we do not support hierarchy deeper than two level */
        if (parent != cgroup->top_cgroup)
                return ERR_PTR(-EPERM);
```

```
> +
> + return 0;
> +}
> +
> +static int tslack_write_range(struct cgroup *cgroup, struct cftype *cft,
> +  u64 val)
```

```
> +{
> + struct timer_slack_cgroup *tslack_cgroup;
> + struct cgroup_iter it;
> + struct task_struct *task;
> + int err;
> +
> + err = validate_change(cgroup, val, cft->private);
> + if (err)
> +  return err;
> +
> + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> + if (cft->private == TIMER_SLACK_MIN)
> +  tslack_cgroup->min_slack_ns = val;
> + else
> +  tslack_cgroup->max_slack_ns = val;
> +
> + /*
> +  * Adjust timer slack value for all tasks in the cgroup to fit
> +  * min-max range.
> +  */
> + cgroup_iter_start(cgroup, &it);
> + while ((task = cgroup_iter_next(cgroup, &it)))
> +  tslack_adjust_task(tslack_cgroup, task);
> + cgroup_iter_end(cgroup, &it);
```

So, you should "adjust" child cgroups too rather than return -EBUSY from validate_change().

```
> +
> + return 0;
> +}
> +
> +static struct cftype files[] = {
> + {
> +  .name = "set_slack_ns",
> +  .write_u64 = tslack_write_set_slack_ns,
> + },
> + {
> +  .name = "min_slack_ns",
> +  .private = TIMER_SLACK_MIN,
> +  .read_u64 = tslack_read_range,
> +  .write_u64 = tslack_write_range,
> + },
> + {
> +  .name = "max_slack_ns",
> +  .private = TIMER_SLACK_MAX,
> +  .read_u64 = tslack_read_range,
> +  .write_u64 = tslack_write_range,
```

> + },

I didn't get a reply on how a max_slack_ns is useful. It seems
prudent to add as little interface as possible and only when
we clearly see the utility of it.

Cheers,
-Matt Helsley

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

---

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Thomas Gleixner on Mon, 14 Feb 2011 14:00:03 GMT
View Forum Message <> Reply to Message

On Mon, 14 Feb 2011, Kirill A. Shutsemov wrote:
> From: Kirill A. Shutemov <kirill@shutemov.name>
>
> Every task_struct has timer_slack_ns value. This value uses to round up
> poll() and select() timeout values. This feature can be useful in
> mobile environment where combined wakeups are desired.
>
> cgroup subsys "timer_slack" implement timer slack controller. It
> provides a way to group tasks by timer slack value and manage the
> value of group's tasks.

I have no objections against the whole thing in general, but why do we
need a module for this? Why can't we add this to the cgroups muck and
compile it in?

> +struct cgroup_subsys timer_slack_subsys;
> +struct timer_slack_cgroup {
> + struct cgroup_subsys_state css;
> + unsigned long min_slack_ns;
> + unsigned long max_slack_ns;
> +};
> +
> +enum {
> + TIMER_SLACK_MIN,
> + TIMER_SLACK_MAX,
> +};
> +
> +static struct timer_slack_cgroup *cgroup_to_tslack_cgroup(struct cgroup *cgroup)
> +{
> + struct cgroup_subsys_state *css;

> +
> + css = cgroup_subsys_state(cgroup, timer_slack_subsys.subsys_id);
> + return container_of(css, struct timer_slack_cgroup, css);
> +}
> +
> +static int is_timer_slack_allowed(struct timer_slack_cgroup *tslack_cgroup,

bool perhaps ?

> + unsigned long slack_ns)
> +{
> + if (slack_ns < tslack_cgroup->min_slack_ns ||
> + slack_ns > tslack_cgroup->max_slack_ns)
> + return false;
> + return true;
> +}
> +
> +static int cgroup_timer_slack_check(struct notifier_block *nb,
> + unsigned long slack_ns, void *data)
> +{
> + struct cgroup_subsys_state *css;
> + struct timer_slack_cgroup *tslack_cgroup;
> +
> + /* XXX: lockdep false positive? */

What? Either this has a reason or not. If it's a false positive then
it needs to be fixed in lockdep. If not, ...

> + rcu_read_lock();
> + css = task_subsys_state(current, timer_slack_subsys.subsys_id);
> + tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
> + rcu_read_unlock();
> +
> + if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
> + return notifier_from_errno(-EPERM);

If the above needs rcu read lock, why is the acess safe ?

> + return NOTIFY_OK;

> +/*
> + * Adjust ->timer_slack_ns and ->default_max_slack_ns of the task to fit
> + * limits of the cgroup.
> + */
> +static void tslack_adjust_task(struct timer_slack_cgroup *tslack_cgroup,
> + struct task_struct *tsk)
> +{
> + if (tslack_cgroup->min_slack_ns > tsk->timer_slack_ns)

> + tsk->timer_slack_ns = tslack_cgroup->min_slack_ns;
> + else if (tslack_cgroup->max_slack_ns < tsk->timer_slack_ns)
> + tsk->timer_slack_ns = tslack_cgroup->max_slack_ns;
> +
> + if (tslack_cgroup->min_slack_ns > tsk->default_timer_slack_ns)
> + tsk->default_timer_slack_ns = tslack_cgroup->min_slack_ns;
> + else if (tslack_cgroup->max_slack_ns < tsk->default_timer_slack_ns)
> + tsk->default_timer_slack_ns = tslack_cgroup->max_slack_ns;

Why is there not a default slack value for the whole group ?

> +static u64 tslack_read_range(struct cgroup *cgroup, struct cftype *cft)
> +{
> + struct timer_slack_cgroup *tslack_cgroup;
> +
> + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> + switch (cft->private) {
> + case TIMER_SLACK_MIN:
> + return tslack_cgroup->min_slack_ns;
> + case TIMER_SLACK_MAX:
> + return tslack_cgroup->max_slack_ns;
> + default:
> + BUG();

BUG() for soemthing which can be dealt with sensible ?

> + }
> +}
> +
> +static int validate_change(struct cgroup *cgroup, u64 val, int type)
> +{
> + struct timer_slack_cgroup *tslack_cgroup, *child;
> + struct cgroup *cur;
> +
> + BUG_ON(type != TIMER_SLACK_MIN && type != TIMER_SLACK_MAX);

Ditto. That should be -EINVAL or such.

> + if (val > ULONG_MAX)
> + return -EINVAL;
> +
> + if (cgroup->parent) {
> + struct timer_slack_cgroup *parent;
> + parent = cgroup_to_tslack_cgroup(cgroup->parent);
> + if (!is_timer_slack_allowed(parent, val))
> +   return -EPERM;
> + }

> +
> + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> + if (type == TIMER_SLACK_MIN && val > tslack_cgroup->max_slack_ns)
> + return -EINVAL;
> + if (type == TIMER_SLACK_MAX && val < tslack_cgroup->min_slack_ns)
> + return -EINVAL;
> +
> + list_for_each_entry(cur, &cgroup->children, sibling) {
> + child = cgroup_to_tslack_cgroup(cur);
> + if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
> +   return -EBUSY;

I thought the whole point is to propagate values through the group.

> + if (type == TIMER_SLACK_MAX && val < child->max_slack_ns)
> +   return -EBUSY;

This is completely confusing w/o any line of comment.

Thanks

tglx

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

---

Subject: Re: [PATCH, v6 2/3] Implement timer slack notifier chain
Posted by Kirill A. Shutsemov on Mon, 14 Feb 2011 14:52:44 GMT
View Forum Message <> Reply to Message

On Mon, Feb 14, 2011 at 02:32:23PM +0100, Thomas Gleixner wrote:
> On Mon, 14 Feb 2011, Kirill A. Shutsemov wrote:
>
> > From: Kirill A. Shutemov <kirill@shutemov.name>
> >
> > Process can change its timer slack using prctl(). Timer slack notifier
> > call chain allows to react on such change or forbid it.
>
> So we add a notifier call chain and more exports to allow what ?

To allow the cgroup contoller validate the value.

> > --- a/kernel/sys.c
> > +++ b/kernel/sys.c
> > @@ -1691,15 +1691,10 @@ SYSCALL_DEFINE5(prctl, int, option, unsigned long, arg2,
unsigned long, arg3,

> >   error = perf_event_task_enable();
> >   break;
> >   case PR_GET_TIMERSLACK:
> > -   error = current->timer_slack_ns;
> > +   error = prctl_get_timer_slack();
>
>   What's the point of replacing current->timer_slack_ns with a
>   function which does exactly the same ?

To keep it consistent. BTW, prctl_get_seccomp() does the same.

>
> > +long prctl_set_timer_slack(long timer_slack_ns)
> > +{
> > + int err;
> > +
> > + /* Reset timer slack to default value */
> > + if (timer_slack_ns <= 0) {
> > + current->timer_slack_ns = current->default_timer_slack_ns;
> > + return 0;
>
>   That does not make any sense at all. Why is setting
>   default_timer_slack_ns not subject to validation ?

Hm.. In case of cgroup_timer_slack it's always valid.
But, yes, in general, we should validate it.

>   Why is it treaded seperately ?

What do you mean?

> > + }
> > +
> > + err = blocking_notifier_call_chain(&timer_slack_notify_list,
> > +   timer_slack_ns, NULL);
> > + if (err == NOTIFY_DONE)
> > + current->timer_slack_ns = timer_slack_ns;
> > +
> > + return notifier_to_errno(err);
>
> Thanks,
>
> tglx

--
 Kirill A. Shutemov

_____
Containers mailing list

Subject: Re: [PATCH, v6 2/3] Implement timer slack notifier chain
Posted by Thomas Gleixner on Mon, 14 Feb 2011 15:16:28 GMT
View Forum Message <> Reply to Message

On Mon, 14 Feb 2011, Kirill A. Shutemov wrote:
> On Mon, Feb 14, 2011 at 02:32:23PM +0100, Thomas Gleixner wrote:
> > On Mon, 14 Feb 2011, Kirill A. Shutsemov wrote:
> >
> > > From: Kirill A. Shutemov <kirill@shutemov.name>
> > >
> > > Process can change its timer slack using prctl(). Timer slack notifier
> > > call chain allows to react on such change or forbid it.
> >
> > So we add a notifier call chain and more exports to allow what ?
>
> To allow the cgroup contoller validate the value.

So we add 5 exports and a notifier chain to have a module? Errm, I
mean there is not really a high probability that we'll add 5 more of
those validation thingies, right?

So instead of having
#ifdef CONFIG_CGROUP_MUCK
int cgroup_set_slack(....);
#else
static inline int cgroup_set_slack(...)
{
 return ....
}
#endif

We add all that stuff ?

> > > --- a/kernel/sys.c
> > > +++ b/kernel/sys.c
> > > @@ -1691,15 +1691,10 @@ SYSCALL_DEFINE5(prctl, int, option, unsigned long, arg2,
unsigned long, arg3,
> > > error = perf_event_task_enable();
> > > break;
> > > case PR_GET_TIMERSLACK:
> > > - error = current->timer_slack_ns;
> > > + error = prctl_get_timer_slack();
> >
> > What's the point of replacing current->timer_slack_ns with a

> > function which does exactly the same ?
>
> To keep it consistent. BTW, prctl_get_seccomp() does the same.

That does not make it less bloat.

> >
> > > +long prctl_set_timer_slack(long timer_slack_ns)
> > > +{
> > > + int err;
> > > +
> > > + /* Reset timer slack to default value */
> > > + if (timer_slack_ns <= 0) {
> > > + current->timer_slack_ns = current->default_timer_slack_ns;
> > > + return 0;
> > >
> > That does not make any sense at all. Why is setting
> > default_timer_slack_ns not subject to validation ?
>
> Hm.. In case of cgroup_timer_slack it's always valid.
> But, yes, in general, we should validate it.
>
> > Why is it treaded seperately ?
>
> What do you mean?

Should have read:

Why is it treated seperately from the other settings?

So setting the default is probably correct to be out of the validation thing, still the question remains, why we do not have a cgroup default then.

Thanks,

tglx

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

---

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Kirill A. Shutsemov on Mon, 14 Feb 2011 15:19:14 GMT
View Forum Message <> Reply to Message

On Mon, Feb 14, 2011 at 03:00:03PM +0100, Thomas Gleixner wrote:

> On Mon, 14 Feb 2011, Kirill A. Shutsemov wrote:
> > From: Kirill A. Shutemov <kirill@shutemov.name>
> >
> > Every task_struct has timer_slack_ns value. This value uses to round up
> > poll() and select() timeout values. This feature can be useful in
> > mobile environment where combined wakeups are desired.
> >
> > cgroup subsys "timer_slack" implement timer slack controller. It
> > provides a way to group tasks by timer slack value and manage the
> > value of group's tasks.
>
> I have no objections against the whole thing in general, but why do we
> need a module for this? Why can't we add this to the cgroups muck and
> compile it in?

It was easier to test and debug with module.
What is wrong with module? Do you worry about number of exports?

> > +struct cgroup_subsys timer_slack_subsys;
> > +struct timer_slack_cgroup {
> > + struct cgroup_subsys_state css;
> > + unsigned long min_slack_ns;
> > + unsigned long max_slack_ns;
> > +};
> > +
> > +enum {
> > + TIMER_SLACK_MIN,
> > + TIMER_SLACK_MAX,
> > +};
> > +
> > +static struct timer_slack_cgroup *cgroup_to_tslack_cgroup(struct cgroup *cgroup)
> > +{
> > + struct cgroup_subsys_state *css;
> > +
> > + css = cgroup_subsys_state(cgroup, timer_slack_subsys.subsys_id);
> > + return container_of(css, struct timer_slack_cgroup, css);
> > +}
> > +
> > +static int is_timer_slack_allowed(struct timer_slack_cgroup *tslack_cgroup,
>
> bool perhaps ?

Right.

> > + unsigned long slack_ns)
> > +{
> > + if (slack_ns < tslack_cgroup->min_slack_ns ||
> > + slack_ns > tslack_cgroup->max_slack_ns)

> > + return false;
> > + return true;
> > +}
> > +
> > +static int cgroup_timer_slack_check(struct notifier_block *nb,
> > + unsigned long slack_ns, void *data)
> > +{
> > + struct cgroup_subsys_state *css;
> > + struct timer_slack_cgroup *tslack_cgroup;
> > +
> > + /* XXX: lockdep false positive? */
>
> What? Either this has a reason or not. If it's a false positive then
> it needs to be fixed in lockdep. If not, ...

I was not sure about it. There is similar workaround in freezer_fork().

> > + rcu_read_lock();
> > + css = task_subsys_state(current, timer_slack_subsys.subsys_id);
> > + tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
> > + rcu_read_unlock();
> > +
> > + if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
> > + return notifier_from_errno(-EPERM);
>
> If the above needs rcu read lock, why is the acess safe ?
>
> > + return NOTIFY_OK;
>
> > +/*
> > + * Adjust ->timer_slack_ns and ->default_max_slack_ns of the task to fit
> > + * limits of the cgroup.
> > + */
> > +static void tslack_adjust_task(struct timer_slack_cgroup *tslack_cgroup,
> > + struct task_struct *tsk)
> > +{
> > + if (tslack_cgroup->min_slack_ns > tsk->timer_slack_ns)
> > + tsk->timer_slack_ns = tslack_cgroup->min_slack_ns;
> > + else if (tslack_cgroup->max_slack_ns < tsk->timer_slack_ns)
> > + tsk->timer_slack_ns = tslack_cgroup->max_slack_ns;
> > +
> > + if (tslack_cgroup->min_slack_ns > tsk->default_timer_slack_ns)
> > + tsk->default_timer_slack_ns = tslack_cgroup->min_slack_ns;
> > + else if (tslack_cgroup->max_slack_ns < tsk->default_timer_slack_ns)
> > + tsk->default_timer_slack_ns = tslack_cgroup->max_slack_ns;
>
>
> Why is there not a default slack value for the whole group ?

I think it breaks prctl() semantic. default slack value is a value on
fork().

> > +static u64 tslack_read_range(struct cgroup *cgroup, struct cftype *cft)
> > +{
> > + struct timer_slack_cgroup *tslack_cgroup;
> > +
> > + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> > + switch (cft->private) {
> > + case TIMER_SLACK_MIN:
> > + return tslack_cgroup->min_slack_ns;
> > + case TIMER_SLACK_MAX:
> > + return tslack_cgroup->max_slack_ns;
> > + default:
> > + BUG();
>
> BUG() for soemthing which can be dealt with sensible ?

tslack_read_range() and tslack_write_range() have written to handle
defined cftypes. If it used for other cftype it's a bug().

> > + }
> > +}
> > +
> > +static int validate_change(struct cgroup *cgroup, u64 val, int type)
> > +{
> > + struct timer_slack_cgroup *tslack_cgroup, *child;
> > + struct cgroup *cur;
> > +
> > + BUG_ON(type != TIMER_SLACK_MIN && type != TIMER_SLACK_MAX);
>
> Ditto. That should be -EINVAL or such.
>
> > + if (val > ULONG_MAX)
> > + return -EINVAL;
> > +
> > + if (cgroup->parent) {
> > + struct timer_slack_cgroup *parent;
> > + parent = cgroup_to_tslack_cgroup(cgroup->parent);
> > + if (!is_timer_slack_allowed(parent, val))
> > + return -EPERM;
> > + }
> > +
> > + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> > + if (type == TIMER_SLACK_MIN && val > tslack_cgroup->max_slack_ns)
> > + return -EINVAL;
> > + if (type == TIMER_SLACK_MAX && val < tslack_cgroup->min_slack_ns)

> > + return -EINVAL;
> > +
> > + list_for_each_entry(cur, &cgroup->children, sibling) {
> > + child = cgroup_to_tslack_cgroup(cur);
> > + if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
> > + return -EBUSY;
>
> I thought the whole point is to propagate values through the group.

I think silent change here is wrong. cpuset returns -EBUSY in similar
case.

> > + if (type == TIMER_SLACK_MAX && val < child->max_slack_ns)
> > + return -EBUSY;
>
> This is completely confusing w/o any line of comment.

Ok, I'll add a comment here.
>
> Thanks
>
> tglx

--
Kirill A. Shutemov

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

---

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Thomas Gleixner on Mon, 14 Feb 2011 17:01:06 GMT
View Forum Message <> Reply to Message

B1;2401;0cOn Mon, 14 Feb 2011, Kirill A. Shutemov wrote:

> On Mon, Feb 14, 2011 at 03:00:03PM +0100, Thomas Gleixner wrote:
> > On Mon, 14 Feb 2011, Kirill A. Shutsemov wrote:
> > > From: Kirill A. Shutemov <kirill@shutemov.name>
> > >
> > > Every task_struct has timer_slack_ns value. This value uses to round up
> > > poll() and select() timeout values. This feature can be useful in
> > > mobile environment where combined wakeups are desired.
> > >
> > > cgroup subsys "timer_slack" implement timer slack controller. It
> > > provides a way to group tasks by timer slack value and manage the
> > > value of group's tasks.

> >
> > I have no objections against the whole thing in general, but why do we
> > need a module for this? Why can't we add this to the cgroups muck and
> > compile it in?
>
> It was easier to test and debug with module.
> What is wrong with module? Do you worry about number of exports?

Not only about the number. We don't want exports when they are not
techically necessary, i.e. for driver stuff.

> > > +static int cgroup_timer_slack_check(struct notifier_block *nb,
> > > + unsigned long slack_ns, void *data)
> > > +{
> > > + struct cgroup_subsys_state *css;
> > > + struct timer_slack_cgroup *tslack_cgroup;
> > > +
> > > + /* XXX: lockdep false positive? */
> >
> >   What? Either this has a reason or not. If it's a false positive then
> >   it needs to be fixed in lockdep. If not, ...
>
> I was not sure about it. There is similar workaround in freezer_fork().

I don't care about workarounds in freezer_work() at all. The above
question remains and this is new code and therefor it either needs to
hold rcu_read_lock() or it does not.

> > > + rcu_read_lock();
> > > + css = task_subsys_state(current, timer_slack_subsys.subsys_id);
> > > + tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
> > > + rcu_read_unlock();
> > > +
> > > + if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
> > > + return notifier_from_errno(-EPERM);
> >
> >   If the above needs rcu read lock, why is the acess safe ?
> >
> > > + return NOTIFY_OK;
> >
> > > +/*
> > > + * Adjust ->timer_slack_ns and ->default_max_slack_ns of the task to fit
> > > + * limits of the cgroup.
> > > + */
> > > +static void tslack_adjust_task(struct timer_slack_cgroup *tslack_cgroup,
> > > + struct task_struct *tsk)
> > > +{
> > > + if (tslack_cgroup->min_slack_ns > tsk->timer_slack_ns)

> > > + tsk->timer_slack_ns = tslack_cgroup->min_slack_ns;
> > > + else if (tslack_cgroup->max_slack_ns < tsk->timer_slack_ns)
> > > + tsk->timer_slack_ns = tslack_cgroup->max_slack_ns;
> > > +
> > > + if (tslack_cgroup->min_slack_ns > tsk->default_timer_slack_ns)
> > > + tsk->default_timer_slack_ns = tslack_cgroup->min_slack_ns;
> > > + else if (tslack_cgroup->max_slack_ns < tsk->default_timer_slack_ns)
> > > + tsk->default_timer_slack_ns = tslack_cgroup->max_slack_ns;
> >
> >
> >   Why is there not a default slack value for the whole group ?
>
> I think it breaks prctl() semantic. default slack value is a value on
> fork().

cgroups break a lot of semantics.

> > > +static u64 tslack_read_range(struct cgroup *cgroup, struct cftype *cft)
> > > +{
> > > + struct timer_slack_cgroup *tslack_cgroup;
> > > +
> > > + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> > > + switch (cft->private) {
> > > + case TIMER_SLACK_MIN:
> > > + return tslack_cgroup->min_slack_ns;
> > > + case TIMER_SLACK_MAX:
> > > + return tslack_cgroup->max_slack_ns;
> > > + default:
> > > + BUG();
> >
> >   BUG() for soemthing which can be dealt with sensible ?
>
> tslack_read_range() and tslack_write_range() have written to handle
> defined cftypes. If it used for other cftype it's a bug().

The only caller is initiated from here, right? So we really don't need
another bug just because you might fatfinger your own code.

> > > + list_for_each_entry(cur, &cgroup->children, sibling) {
> > > + child = cgroup_to_tslack_cgroup(cur);
> > > + if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
> > > +   return -EBUSY;
> >
> >   I thought the whole point is to propagate values through the group.
>
> I think silent change here is wrong. cpuset returns -EBUSY in similar
> case.

And how is cpuset relevant for this ? Not at all. This is about
timer_slack and we better have a well defined scheme for all of this
and not some cobbled together thing with tons of exceptions and corner
cases. Of course undocumented as far the code goes.

Thanks,

tglx

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

---

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Kirill A. Shutsemov on Mon, 14 Feb 2011 22:39:39 GMT
View Forum Message <> Reply to Message

On Mon, Feb 14, 2011 at 06:01:06PM +0100, Thomas Gleixner wrote:
> B1;2401;0cOn Mon, 14 Feb 2011, Kirill A. Shutemov wrote:
>
> > On Mon, Feb 14, 2011 at 03:00:03PM +0100, Thomas Gleixner wrote:
> > > On Mon, 14 Feb 2011, Kirill A. Shutsemov wrote:
> > > > From: Kirill A. Shutemov <kirill@shutemov.name>
> > > >
> > > > Every task_struct has timer_slack_ns value. This value uses to round up
> > > > poll() and select() timeout values. This feature can be useful in
> > > > mobile environment where combined wakeups are desired.
> > > >
> > > > cgroup subsys "timer_slack" implement timer slack controller. It
> > > > provides a way to group tasks by timer slack value and manage the
> > > > value of group's tasks.
> > >
> > > I have no objections against the whole thing in general, but why do we
> > > need a module for this? Why can't we add this to the cgroups muck and
> > > compile it in?
> >
> > It was easier to test and debug with module.
> > What is wrong with module? Do you worry about number of exports?
>
> Not only about the number. We don't want exports when they are not
> techically necessary, i.e. for driver stuff.

Ok, I'll drop module support.

> > > > +static int cgroup_timer_slack_check(struct notifier_block *nb,

> > > > + unsigned long slack_ns, void *data)
> > > > +{
> > > > + struct cgroup_subsys_state *css;
> > > > + struct timer_slack_cgroup *tslack_cgroup;
> > > > +
> > > > + /* XXX: lockdep false positive? */
> > >
> > > What? Either this has a reason or not. If it's a false positive then
> > > it needs to be fixed in lockdep. If not, ...
> >
> > I was not sure about it. There is similar workaround in freezer_fork().
>
> I don't care about workarounds in freezer_work() at all. The above
> question remains and this is new code and therefor it either needs to
> hold rcu_read_lock() or it does not.

I'll recheck everything once again.

> > > > + rcu_read_lock();
> > > > + css = task_subsys_state(current, timer_slack_subsys.subsys_id);
> > > > + tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
> > > > + rcu_read_unlock();
> > > > +
> > > > + if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
> > > > + return notifier_from_errno(-EPERM);
> > >
> > > If the above needs rcu read lock, why is the acess safe ?
> > >
> > > > + return NOTIFY_OK;
> > >
> > > > +/*
> > > > + * Adjust ->timer_slack_ns and ->default_max_slack_ns of the task to fit
> > > > + * limits of the cgroup.
> > > > + */
> > > > +static void tslack_adjust_task(struct timer_slack_cgroup *tslack_cgroup,
> > > > + struct task_struct *tsk)
> > > > +{
> > > > + if (tslack_cgroup->min_slack_ns > tsk->timer_slack_ns)
> > > > + tsk->timer_slack_ns = tslack_cgroup->min_slack_ns;
> > > > + else if (tslack_cgroup->max_slack_ns < tsk->timer_slack_ns)
> > > > + tsk->timer_slack_ns = tslack_cgroup->max_slack_ns;
> > > > +
> > > > + if (tslack_cgroup->min_slack_ns > tsk->default_timer_slack_ns)
> > > > + tsk->default_timer_slack_ns = tslack_cgroup->min_slack_ns;
> > > > + else if (tslack_cgroup->max_slack_ns < tsk->default_timer_slack_ns)
> > > > + tsk->default_timer_slack_ns = tslack_cgroup->max_slack_ns;
> > >
> > >

> > > Why is there not a default slack value for the whole group ?
> >
> > I think it breaks prctl() semantic. default slack value is a value on
> > fork().
>
> cgroups break a lot of semantics.

I don't know what "a lot of semantics" you mean, but it's not a reason
to add more breakage.

> > > > +static u64 tslack_read_range(struct cgroup *cgroup, struct cftype *cft)
> > > > +{
> > > > + struct timer_slack_cgroup *tslack_cgroup;
> > > > +
> > > > + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> > > > + switch (cft->private) {
> > > > + case TIMER_SLACK_MIN:
> > > > + return tslack_cgroup->min_slack_ns;
> > > > + case TIMER_SLACK_MAX:
> > > > + return tslack_cgroup->max_slack_ns;
> > > > + default:
> > > > + BUG();
> > >
> > > BUG() for soemthing which can be dealt with sensible ?
> >
> > tslack_read_range() and tslack_write_range() have written to handle
> > defined cftypes. If it used for other cftype it's a bug().
>
> The only caller is initiated from here, right? So we really don't need
> another bug just because you might fatfinger your own code.

People make mistakes. I think BUG() is useful here.

> > > > + list_for_each_entry(cur, &cgroup->children, sibling) {
> > > > + child = cgroup_to_tslack_cgroup(cur);
> > > > + if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
> > > > +   return -EBUSY;
> > >
> > > I thought the whole point is to propagate values through the group.
> >
> > I think silent change here is wrong. cpuset returns -EBUSY in similar
> > case.
>
> And how is cpuset relevant for this ? Not at all. This is about
> timer_slack and we better have a well defined scheme for all of this
> and not some cobbled together thing with tons of exceptions and corner
> cases. Of course undocumented as far the code goes.

I don't like silent cascade changes. Userspace can implement it if
needed. -EBUSY is appropriate.

--
Kirill A. Shutemov

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

---

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Kirill A. Shutsemov on Mon, 14 Feb 2011 22:59:40 GMT
View Forum Message <> Reply to Message

On Mon, Feb 14, 2011 at 05:59:26AM -0800, Matt Helsley wrote:
> On Mon, Feb 14, 2011 at 03:06:27PM +0200, Kirill A. Shutsemov wrote:
> > From: Kirill A. Shutemov <kirill@shutemov.name>
> >
> > Every task_struct has timer_slack_ns value. This value uses to round up
> > poll() and select() timeout values. This feature can be useful in
> > mobile environment where combined wakeups are desired.
> >
> > cgroup subsys "timer_slack" implement timer slack controller. It
> > provides a way to group tasks by timer slack value and manage the
> > value of group's tasks.
> >
> > Idea-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
> > Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
> > ---
> > Documentation/cgroups/timer_slack.txt | 93 ++++++++++++
> > include/linux/cgroup_subsys.h | 6 +
> > init/Kconfig | 10 ++
> > kernel/Makefile | 1 +
> > kernel/cgroup_timer_slack.c | 285 +++++++++++++++++++++++++++++++++
> > 5 files changed, 395 insertions(+), 0 deletions(-)
> > create mode 100644 Documentation/cgroups/timer_slack.txt
> > create mode 100644 kernel/cgroup_timer_slack.c
> >
> > diff --git a/Documentation/cgroups/timer_slack.txt b/Documentation/cgroups/timer_slack.txt
> > new file mode 100644
> > index 0000000..e7ec2f3
> > --- /dev/null
> > +++ b/Documentation/cgroups/timer_slack.txt
> > @@ -0,0 +1,93 @@
> > +Timer Slack Controller
> > +======================
> > +

> > +Overview
> > +--------
> > +
> > +Every task_struct has timer_slack_ns value. This value uses to round up
> > +poll() and select() timeout values. This feature can be useful in
> > +mobile environment where combined wakeups are desired.
> > +
> > +cgroup subsys "timer_slack" implement timer slack controller. It
> > +provides a way to group tasks by timer slack value and manage the
> > +value of group's tasks.
> > +
> > +
> > +User interface
> > +--------------
> > +
> > +To get timer slack controller functionality you need to enable it in
> > +kernel configuration:
> > +
> > +CONFIG_CGROUP_TIMER_SLACK=y
> > +
> > +or if you want to compile it as module:
> > +
> > +CONFIG_CGROUP_TIMER_SLACK=m
> > +
> > +The controller provides three files in cgroup directory:
> > +
> > +# mount -t cgroup -o timer_slack none /sys/fs/cgroup
> > +# ls /sys/fs/cgroup/timer_slack.*
> > +/sys/fs/cgroup/timer_slack.max_slack_ns
> > +/sys/fs/cgroup/timer_slack.min_slack_ns
> > +/sys/fs/cgroup/timer_slack.set_slack_ns
> > +
> > +timer_slack.min_slack_ns and timer_slack.set_slack_ns specify allowed
> > +range of timer slack for tasks in cgroup. By default it unlimited:
> > +
> > +# cat /sys/fs/cgroup/timer_slack.min_slack_ns
> > +0
> > +# cat /sys/fs/cgroup/timer_slack.max_slack_ns
> > +4294967295
> > +
> > +You can specify limits you want:
> > +
> > +# echo 50000 > /sys/fs/cgroup/timer_slack.min_slack_ns
> > +# echo 1000000 > /sys/fs/cgroup/timer_slack.max_slack_ns
> > +# cat /sys/fs/cgroup/timer_slack.{min,max}_slack_ns
> > +50000
> > +1000000
> > +

> > +Timer slack value of all tasks of the cgroup will be adjusted to fit
> > +min-max range.
> > +
> > +If a task will try to call prctl() to change timer slack value out of
> > +the range it get -EPERM.
> > +
> > +You can change timer slack value of all tasks of the cgroup at once:
> > +
> > +# echo 70000 > /sys/fs/cgroup/timer_slack.set_slack_ns
> > +
> > +Timer slack controller supports hierarchical groups. The only rule:
> > +parent's limit range should be wider or equal to child's. Sibling
> > +cgroups can have overlapping min-max range.
> > +
> > + (root: 50000 - 1000000)
> > +   /   \
> > + (a: 50000 - 50000) (b: 500000 - 1000000)
> > +   /   \
> > +   (c: 500000 - 900000) (d: 700000 - 800000)
> > +
> > +# mkdir /sys/fs/cgroup/a
> > +# echo 50000 > /sys/fs/cgroup/a/timer_slack.max_slack_ns
> > +# cat /sys/fs/cgroup/a/timer_slack.{min,max}_slack_ns
> > +50000
> > +50000
> > +# mkdir /sys/fs/cgroup/b
> > +# echo 500000 > /sys/fs/cgroup/b/timer_slack.min_slack_ns
> > +# cat /sys/fs/cgroup/b/timer_slack.{min,max}_slack_ns
> > +500000
> > +1000000
> > +# mkdir /sys/fs/cgroup/b/c
> > +# echo 900000 > /sys/fs/cgroup/b/c/timer_slack.max_slack_ns
> > +# cat /sys/fs/cgroup/b/c/timer_slack.{min,max}_slack_ns
> > +500000
> > +900000
> > +# mkdir /sys/fs/cgroup/b/d
> > +# echo 700000 > /sys/fs/cgroup/b/d/timer_slack.min_slack_ns
> > +# echo 800000 > /sys/fs/cgroup/b/d/timer_slack.max_slack_ns
> > +# cat /sys/fs/cgroup/b/d/timer_slack.{min,max}_slack_ns
> > +700000
> > +800000
> > +
> > diff --git a/include/linux/cgroup_subsys.h b/include/linux/cgroup_subsys.h
> > index ccefff0..e399228 100644
> > --- a/include/linux/cgroup_subsys.h
> > +++ b/include/linux/cgroup_subsys.h
> > @@ -66,3 +66,9 @@ SUBSYS(blkio)
> > #endif

> >
> > /* */
> > +
> > +#ifdef CONFIG_CGROUP_TIMER_SLACK
> > +SUBSYS(timer_slack)
> > +#endif
> > +
> > +/* */
> > diff --git a/init/Kconfig b/init/Kconfig
> > index be788c0..bb54ae9 100644
> > --- a/init/Kconfig
> > +++ b/init/Kconfig
> > @@ -596,6 +596,16 @@ config CGROUP_FREEZER
> >   Provides a way to freeze and unfreeze all tasks in a
> >   cgroup.
> >
> > +config CGROUP_TIMER_SLACK
> > + tristate "Timer slack cgroup controller"
> > + help
> > +   Provides a way of tasks grouping by timer slack value.
> > +   Every timer slack cgroup has min and max slack value. Task's
> > +   timer slack value will be adjusted to fit min-max range once
> > +   the task attached to the cgroup.
> > +   It's useful in mobile devices where certain background apps
> > +   are attached to a cgroup and combined wakeups are desired.
> > +
> > config CGROUP_DEVICE
> >   bool "Device controller for cgroups"
> >   help
> > diff --git a/kernel/Makefile b/kernel/Makefile
> > index aa43e63..0e660c5 100644
> > --- a/kernel/Makefile
> > +++ b/kernel/Makefile
> > @@ -61,6 +61,7 @@ obj-$(CONFIG_BACKTRACE_SELF_TEST) += backtracetest.o
> > obj-$(CONFIG_COMPAT) += compat.o
> > obj-$(CONFIG_CGROUPS) += cgroup.o
> > obj-$(CONFIG_CGROUP_FREEZER) += cgroup_freezer.o
> > +obj-$(CONFIG_CGROUP_TIMER_SLACK) += cgroup_timer_slack.o
> > obj-$(CONFIG_CPUSETS) += cpuset.o
> > obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
> > obj-$(CONFIG_UTS_NS) += utsname.o
> > diff --git a/kernel/cgroup_timer_slack.c b/kernel/cgroup_timer_slack.c
> > new file mode 100644
> > index 0000000..ec63700
> > --- /dev/null
> > +++ b/kernel/cgroup_timer_slack.c
> > @@ -0,0 +1,285 @@
> > +/*

```
> > + * cgroup_timer_slack.c - control group timer slack subsystem
> > + *
> > + * Copyright Nokia Corparation, 2011
> > + * Author: Kirill A. Shutemov
> > + *
> > + * This program is free software; you can redistribute it and/or modify
> > + * it under the terms of the GNU General Public License as published by
> > + * the Free Software Foundation; either version 2 of the License, or
> > + * (at your option) any later version.
> > + *
> > + * This program is distributed in the hope that it will be useful,
> > + * but WITHOUT ANY WARRANTY; without even the implied warranty of
> > + * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
> > + * GNU General Public License for more details.
> > + */
> > +#include <linux/cgroup.h>
> > +#include <linux/init_task.h>
> > +#include <linux/module.h>
> > +#include <linux/slab.h>
> > +#include <linux/rcupdate.h>
> > +
> > +struct cgroup_subsys timer_slack_subsys;
> > +struct timer_slack_cgroup {
> > + struct cgroup_subsys_state css;
> > + unsigned long min_slack_ns;
> > + unsigned long max_slack_ns;
> > +};
> > +
> > +enum {
> > + TIMER_SLACK_MIN,
> > + TIMER_SLACK_MAX,
> > +};
> > +
> > +static struct timer_slack_cgroup *cgroup_to_tslack_cgroup(struct cgroup *cgroup)
> > +{
> > + struct cgroup_subsys_state *css;
> > +
> > + css = cgroup_subsys_state(cgroup, timer_slack_subsys.subsys_id);
> > + return container_of(css, struct timer_slack_cgroup, css);
> > +}
> > +
> > +static int is_timer_slack_allowed(struct timer_slack_cgroup *tslack_cgroup,
> > +  unsigned long slack_ns)
> > +{
> > + if (slack_ns < tslack_cgroup->min_slack_ns ||
> > +   slack_ns > tslack_cgroup->max_slack_ns)
> > +  return false;
> > + return true;
```

> > +}
> > +
> > +static int cgroup_timer_slack_check(struct notifier_block *nb,
> > + unsigned long slack_ns, void *data)
> > +{
> > + struct cgroup_subsys_state *css;
> > + struct timer_slack_cgroup *tslack_cgroup;
> > +
> > + /* XXX: lockdep false positive? */
> > + rcu_read_lock();
> > + css = task_subsys_state(current, timer_slack_subsys.subsys_id);
> > + tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
> > + rcu_read_unlock();
> > +
> > + if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
>
> I think this test -- or at least accesses to the cgroup's min/max values
> -- needs to be in the rcu_read_lock() else there is a race between the
> notifier call from the context of the task calling prctl() and the context
> of the task writing to the cgroup's (min|max)_slack_ns files.

I'll rework this code.

> > + return notifier_from_errno(-EPERM);
> > + return NOTIFY_OK;
> > +}
> > +
> > +static struct notifier_block cgroup_timer_slack_nb = {
> > + .notifier_call = cgroup_timer_slack_check,
> > +};
> > +
> > +static struct cgroup_subsys_state *
> > +tslack_cgroup_create(struct cgroup_subsys *subsys, struct cgroup *cgroup)
> > +{
> > + struct timer_slack_cgroup *tslack_cgroup;
> > +
> > + tslack_cgroup = kmalloc(sizeof(*tslack_cgroup), GFP_KERNEL);
> > + if (!tslack_cgroup)
> > + return ERR_PTR(-ENOMEM);
> > +
> > + if (cgroup->parent) {
> > + struct timer_slack_cgroup *parent;
> > + parent = cgroup_to_tslack_cgroup(cgroup->parent);
> > + tslack_cgroup->min_slack_ns = parent->min_slack_ns;
> > + tslack_cgroup->max_slack_ns = parent->max_slack_ns;
> > + } else {
> > + tslack_cgroup->min_slack_ns = 0UL;
> > + tslack_cgroup->max_slack_ns = ULONG_MAX;

> > + }
> > +
> > + return &tslack_cgroup->css;
> > +}
> > +
> > +static void tslack_cgroup_destroy(struct cgroup_subsys *subsys,
> > + struct cgroup *cgroup)
> > +{
> > + kfree(cgroup_to_tslack_cgroup(cgroup));
> > +}
> > +
> > +/*
> > + * Adjust ->timer_slack_ns and ->default_max_slack_ns of the task to fit
> > + * limits of the cgroup.
> > + */
> > +static void tslack_adjust_task(struct timer_slack_cgroup *tslack_cgroup,
> > + struct task_struct *tsk)
> > +{
> > + if (tslack_cgroup->min_slack_ns > tsk->timer_slack_ns)
> > + tsk->timer_slack_ns = tslack_cgroup->min_slack_ns;
> > + else if (tslack_cgroup->max_slack_ns < tsk->timer_slack_ns)
> > + tsk->timer_slack_ns = tslack_cgroup->max_slack_ns;
>
> It occured to me there's a macro for this in include/linux/kernel.h:
>
> tsk->timer_slack_ns = clamp(tsk->timer_slack_ns,
>   tslack_cgroup->min_slack_ns,
>   tslack_cgroup->max_slack_ns);

Nice. Thank you.

> > +
> > + if (tslack_cgroup->min_slack_ns > tsk->default_timer_slack_ns)
> > + tsk->default_timer_slack_ns = tslack_cgroup->min_slack_ns;
> > + else if (tslack_cgroup->max_slack_ns < tsk->default_timer_slack_ns)
> > + tsk->default_timer_slack_ns = tslack_cgroup->max_slack_ns;
>
> tsk->default_timer_slack_ns = clamp(tsk->default_timer_slack_ns,
>   tslack_cgroup->min_slack_ns,
>   tslack_cgroup->max_slack_ns);
>
> > +}
> > +
> > +static void tslack_cgroup_attach(struct cgroup_subsys *subsys,
> > + struct cgroup *cgroup, struct cgroup *prev,
> > + struct task_struct *tsk, bool threadgroup)
> > +{
> > + tslack_adjust_task(cgroup_to_tslack_cgroup(cgroup), tsk);

```
> > +}
> > +
> > +static int tslack_write_set_slack_ns(struct cgroup *cgroup, struct cftype *cft,
> > +  u64 val)
> > +{
> > + struct timer_slack_cgroup *tslack_cgroup;
> > + struct cgroup_iter it;
> > + struct task_struct *task;
> > +
> > + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> > + if (!is_timer_slack_allowed(cgroup_to_tslack_cgroup(cgroup), val))
> > +  return -EPERM;
> > +
> > + /* Change timer slack value for all tasks in the cgroup */
> > + cgroup_iter_start(cgroup, &it);
> > + while ((task = cgroup_iter_next(cgroup, &it)))
> > +  task->timer_slack_ns = val;
> > + cgroup_iter_end(cgroup, &it);
> > +
> > + return 0;
> > +}
> > +
> > +static u64 tslack_read_range(struct cgroup *cgroup, struct cftype *cft)
> > +{
> > + struct timer_slack_cgroup *tslack_cgroup;
> > +
> > + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> > + switch (cft->private) {
> > + case TIMER_SLACK_MIN:
> > +  return tslack_cgroup->min_slack_ns;
> > + case TIMER_SLACK_MAX:
> > +  return tslack_cgroup->max_slack_ns;
> > + default:
> > +  BUG();
> > + }
> > +}
> > +
> > +static int validate_change(struct cgroup *cgroup, u64 val, int type)
> > +{
> > + struct timer_slack_cgroup *tslack_cgroup, *child;
> > + struct cgroup *cur;
> > +
> > + BUG_ON(type != TIMER_SLACK_MIN && type != TIMER_SLACK_MAX);
> > +
> > + if (val > ULONG_MAX)
> > +  return -EINVAL;
> > +
> > + if (cgroup->parent) {
```

> > + struct timer_slack_cgroup *parent;
> > + parent = cgroup_to_tslack_cgroup(cgroup->parent);
> > + if (!is_timer_slack_allowed(parent, val))
> > +   return -EPERM;
> > + }
> > +
> > + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> > + if (type == TIMER_SLACK_MIN && val > tslack_cgroup->max_slack_ns)
> > + return -EINVAL;
> > + if (type == TIMER_SLACK_MAX && val < tslack_cgroup->min_slack_ns)
> > + return -EINVAL;
> > +
> > + list_for_each_entry(cur, &cgroup->children, sibling) {
> > + child = cgroup_to_tslack_cgroup(cur);
> > + if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
> > +   return -EBUSY;
> > + if (type == TIMER_SLACK_MAX && val < child->max_slack_ns)
> > +   return -EBUSY;
> > + }
>
> This doesn't look right. Child cgroups should not constrain their
> parents. Instead you should allow the change and propagate the
> constraint to the children.

See discussion with Thomas.

> One thing that might make reviewing such changes to these patches easier
> would be to split out the arbitrary-depth hierarchy support into a
> follow-on patch. You can do it like blkio does in the _create() function:
>
>   /* Currently we do not support hierarchy deeper than two level */
>   if (parent != cgroup->top_cgroup)
>   return ERR_PTR(-EPERM);
>

> > +
> > + return 0;
> > +}
> > +
> > +static int tslack_write_range(struct cgroup *cgroup, struct cftype *cft,
> > + u64 val)
> > +{
> > + struct timer_slack_cgroup *tslack_cgroup;
> > + struct cgroup_iter it;
> > + struct task_struct *task;
> > + int err;
> > +
> > + err = validate_change(cgroup, val, cft->private);
> > + if (err)

> > + return err;
> > +
> > + tslack_cgroup = cgroup_to_tslack_cgroup(cgroup);
> > + if (cft->private == TIMER_SLACK_MIN)
> > + tslack_cgroup->min_slack_ns = val;
> > + else
> > + tslack_cgroup->max_slack_ns = val;
> > +
> > + /*
> > + * Adjust timer slack value for all tasks in the cgroup to fit
> > + * min-max range.
> > + */
> > + cgroup_iter_start(cgroup, &it);
> > + while ((task = cgroup_iter_next(cgroup, &it)))
> > + tslack_adjust_task(tslack_cgroup, task);
> > + cgroup_iter_end(cgroup, &it);
>
> So, you should "adjust" child cgroups too rather than return -EBUSY from
> validate_change().

Ditto.

> > +
> > + return 0;
> > +}
> > +
> > +static struct cftype files[] = {
> > + {
> > + .name = "set_slack_ns",
> > + .write_u64 = tslack_write_set_slack_ns,
> > + },
> > + {
> > + .name = "min_slack_ns",
> > + .private = TIMER_SLACK_MIN,
> > + .read_u64 = tslack_read_range,
> > + .write_u64 = tslack_write_range,
> > + },
> > + {
> > + .name = "max_slack_ns",
> > + .private = TIMER_SLACK_MAX,
> > + .read_u64 = tslack_read_range,
> > + .write_u64 = tslack_write_range,
> > + },
>
> I didn't get a reply on how a max_slack_ns is useful. It seems
> prudent to add as little interface as possible and only when
> we clearly see the utility of it.

For example, you can create two groups (excluding root cgroup):

default - timer slack range 50000-50000
relaxed - timer slack range 500000-unlimited.

Now you can drag tasks between these group without need to reset value on
relaxed -> default transition.

--
Kirill A. Shutemov

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

---

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Matt Helsley on Mon, 14 Feb 2011 23:39:36 GMT
View Forum Message <> Reply to Message

On Tue, Feb 15, 2011 at 12:39:39AM +0200, Kirill A. Shutemov wrote:
> On Mon, Feb 14, 2011 at 06:01:06PM +0100, Thomas Gleixner wrote:
> > B1;2401;0cOn Mon, 14 Feb 2011, Kirill A. Shutemov wrote:
> >
> > > On Mon, Feb 14, 2011 at 03:00:03PM +0100, Thomas Gleixner wrote:
> > > > On Mon, 14 Feb 2011, Kirill A. Shutsemov wrote:
> > > > > From: Kirill A. Shutemov <kirill@shutemov.name>

<snip>

> > > > > + list_for_each_entry(cur, &cgroup->children, sibling) {
> > > > > + child = cgroup_to_tslack_cgroup(cur);
> > > > > + if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
> > > > > +   return -EBUSY;
> > > >
> > > > I thought the whole point is to propagate values through the group.
> > >
> > > I think silent change here is wrong. cpuset returns -EBUSY in similar
> > > case.
> >
> > And how is cpuset relevant for this ? Not at all. This is about

I agree with Thomas here -- cpusets aren't relevant.

> > timer_slack and we better have a well defined scheme for all of this
> > and not some cobbled together thing with tons of exceptions and corner
> > cases. Of course undocumented as far the code goes.
>

> I don't like silent cascade changes. Userspace can implement it if

It need not be totally silent. memcg has a "use_hierarchy" flag file.
Alternately, you could punt for now and disable hierarchy somewhat like
blkio does.

> needed. -EBUSY is appropriate.

Hmm, I haven't thought about that method of cascading enough. The important
question to consider is how will the parent cgroup be constrained if the
owner/group of the children is different and thus disallows userspace from
implementing this cascade. I suppose it's consistent with the owner/group
ids but it hardly seems consistent with the "spirit" of using cgroups to
enable things like containers.

Cheers,
-Matt Helsley

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

---

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Matt Helsley on Tue, 15 Feb 2011 00:00:55 GMT
View Forum Message <> Reply to Message

On Tue, Feb 15, 2011 at 12:59:40AM +0200, Kirill A. Shutemov wrote:
> On Mon, Feb 14, 2011 at 05:59:26AM -0800, Matt Helsley wrote:
> > On Mon, Feb 14, 2011 at 03:06:27PM +0200, Kirill A. Shutsemov wrote:
> > > From: Kirill A. Shutemov <kirill@shutemov.name>

<snip>

> > > + list_for_each_entry(cur, &cgroup->children, sibling) {
> > > + child = cgroup_to_tslack_cgroup(cur);
> > > + if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
> > > + return -EBUSY;
> > > + if (type == TIMER_SLACK_MAX && val < child->max_slack_ns)
> > > + return -EBUSY;
> > > + }
> >
> > This doesn't look right. Child cgroups should not constrain their
> > parents. Instead you should allow the change and propagate the
> > constraint to the children.
>
> See discussion with Thomas.

<OK, shifting this topic to that thread>
<snip>

> > > +static struct cftype files[] = {
> > > + {
> > > + .name = "set_slack_ns",
> > > + .write_u64 = tslack_write_set_slack_ns,
> > > + },
> > > + {
> > > + .name = "min_slack_ns",
> > > + .private = TIMER_SLACK_MIN,
> > > + .read_u64 = tslack_read_range,
> > > + .write_u64 = tslack_write_range,
> > > + },
> > > + {
> > > + .name = "max_slack_ns",
> > > + .private = TIMER_SLACK_MAX,
> > > + .read_u64 = tslack_read_range,
> > > + .write_u64 = tslack_write_range,
> > > + },
> >
> > I didn't get a reply on how a max_slack_ns is useful. It seems
> > prudent to add as little interface as possible and only when
> > we clearly see the utility of it.
>
> For example, you can create two groups (excluding root cgroup):
>
> default - timer slack range 50000-50000
> relaxed - timer slack range 500000-unlimited.
>
> Now you can drag tasks between these group without need to reset value on
> relaxed -> default transition.

Perhaps you misunderstood my point.

Yes, I can see that a maximum allows you to do counter-productive/pointless
little tricks like "setting" the timer slack when you move the task. I
just don't get the point of it. Why is setting a maximum timer slack useful?
If anything it seems like it would be quite counterproductive or pointless
at best because limiting the amount of timer slack would not improve
the wakeup situation -- it could easily make it worse. Are there
any negative consequences to allowing timer slacks as large as
userspace requests -- perhaps even up to ULLONG_MAX? If there are none then
why should we bother providing userspace a knob to set and enforce such a
limit?

Cheers,
 -Matt Helsley

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Kirill A. Shutsemov on Tue, 15 Feb 2011 00:10:04 GMT
View Forum Message <> Reply to Message

On Mon, Feb 14, 2011 at 04:00:55PM -0800, Matt Helsley wrote:
> On Tue, Feb 15, 2011 at 12:59:40AM +0200, Kirill A. Shutemov wrote:
> > On Mon, Feb 14, 2011 at 05:59:26AM -0800, Matt Helsley wrote:
> > > On Mon, Feb 14, 2011 at 03:06:27PM +0200, Kirill A. Shutsemov wrote:
> > > > From: Kirill A. Shutemov <kirill@shutemov.name>
>
> <snip>
>
> > > > + list_for_each_entry(cur, &cgroup->children, sibling) {
> > > > + child = cgroup_to_tslack_cgroup(cur);
> > > > + if (type == TIMER_SLACK_MIN && val > child->min_slack_ns)
> > > > +   return -EBUSY;
> > > > + if (type == TIMER_SLACK_MAX && val < child->max_slack_ns)
> > > > +   return -EBUSY;
> > > > + }
> > >
> > > This doesn't look right. Child cgroups should not constrain their
> > > parents. Instead you should allow the change and propagate the
> > > constraint to the children.
> >
> > See discussion with Thomas.
>
> <OK, shifting this topic to that thread>
> <snip>
>
> > > > +static struct cftype files[] = {
> > > > + {
> > > > + .name = "set_slack_ns",
> > > > + .write_u64 = tslack_write_set_slack_ns,
> > > > + },
> > > > + {
> > > > + .name = "min_slack_ns",
> > > > + .private = TIMER_SLACK_MIN,
> > > > + .read_u64 = tslack_read_range,
> > > > + .write_u64 = tslack_write_range,
> > > > + },
> > > > + {
> > > > + .name = "max_slack_ns",

> > > > + .private = TIMER_SLACK_MAX,
> > > > + .read_u64 = tslack_read_range,
> > > > + .write_u64 = tslack_write_range,
> > > > + },
> > >
> > > I didn't get a reply on how a max_slack_ns is useful. It seems
> > > prudent to add as little interface as possible and only when
> > > we clearly see the utility of it.
> >
> > For example, you can create two groups (excluding root cgroup):
> >
> > default - timer slack range 50000-50000
> > relaxed - timer slack range 500000-unlimited.
> >
> > Now you can drag tasks between these group without need to reset value on
> > relaxed -> default transition.
>
> Perhaps you misunderstood my point.
>
> Yes, I can see that a maximum allows you to do counter-productive/pointless
> little tricks like "setting" the timer slack when you move the task. I
> just don't get the point of it. Why is setting a maximum timer slack useful?
> If anything it seems like it would be quite counterproductive or pointless
> *at best* because limiting the amount of timer slack would not improve
> the wakeup situation -- it could easily make it worse. Are there
> *any* negative consequences to allowing timer slacks as large as
> userspace requests -- perhaps even up to ULLONG_MAX? If there are none then
> why should we bother providing userspace a knob to set and enforce such a
> limit?

Could you describe the interface how you see it?

--
Kirill A. Shutemov

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

---

Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller
Posted by Thomas Gleixner on Tue, 15 Feb 2011 06:04:00 GMT
View Forum Message <> Reply to Message

On Tue, 15 Feb 2011, Kirill A. Shutemov wrote:
> On Mon, Feb 14, 2011 at 06:01:06PM +0100, Thomas Gleixner wrote:
> > > I think silent change here is wrong. cpuset returns -EBUSY in similar
> > > case.

> >
> > And how is cpuset relevant for this ? Not at all. This is about
> > timer_slack and we better have a well defined scheme for all of this
> > and not some cobbled together thing with tons of exceptions and corner
> > cases. Of course undocumented as far the code goes.
>
> I don't like silent cascade changes. Userspace can implement it if
> needed. -EBUSY is appropriate.

And I don't like totally uncommented code which follows come cobbled
together completely non obvious rules.

It's not about what you like. It's about getting useful functionality
when we add a new infrastructure like this.

So could you please explain what the rules of updating are, so that a
reviewer has a chance to understand the rationale of all this.

Thanks,

tglx

_____

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