
Subject: Re: [PATCH, v6 3/3] cgroups: introduce timer slack controller

Posted by [Kirill A. Shutemov](#) 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. Shutemov 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 Corporation, 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 occurred 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/containers>
