Subject: [PATCH, v5 0/2] Timer slack cgroup subsystem
Posted by Kirill A. Shutsemov on Mon, 07 Feb 2011 09:46:00 GMT
View Forum Message <> Reply to Message

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

Changelog:
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 (2):
  cgroups: export cgroup_iter_{start,next,end}
  cgroups: introduce timer slack subsystem

 include/linux/cgroup_subsys.h | 6 +
 include/linux/init_task.h | 4 +-
 init/Kconfig | 10 ++
 kernel/Makefile | 1 +
 kernel/cgroup.c | 3 +
 kernel/cgroup_timer_slack.c | 281 ++
 kernel/sys.c | 19 ++-
 7 files changed, 317 insertions(+), 7 deletions(-)
 create mode 100644 kernel/cgroup_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, v5 1/2] cgroups: export cgroup_iter_{start, next, end}

Posted by Kirill A. Shutsemov on Mon, 07 Feb 2011 09:46:01 GMT
View Forum Message <> Reply to Message

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

Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
Acked-by: Paul Menage <menage@google.com>
---
 kernel/cgroup.c | 3 +++
 1 files changed, 3 insertions(+), 0 deletions(-)

```
diff --git a/kernel/cgroup.c b/kernel/cgroup.c
index b24d702..8234daa 100644
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -2443,6 +2443,7 @@ void cgroup_iter_start(struct cgroup *cgrp, struct cgroup_iter *it)
  it->cg_link = &cgrp->css_sets;
  cgroup_advance_iter(cgrp, it);
 }
+EXPORT_SYMBOL_GPL(cgroup_iter_start);

 struct task_struct *cgroup_iter_next(struct cgroup *cgrp,
      struct cgroup_iter *it)
@@ -2467,11 +2468,13 @@ struct task_struct *cgroup_iter_next(struct cgroup *cgrp,
  }
  return res;
 }
+EXPORT_SYMBOL_GPL(cgroup_iter_next);

 void cgroup_iter_end(struct cgroup *cgrp, struct cgroup_iter *it)
 {
  read_unlock(&css_set_lock);
 }
+EXPORT_SYMBOL_GPL(cgroup_iter_end);

 static inline int started_after_time(struct task_struct *t1,
         struct timespec *time,
--
1.7.4
```

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

Subject: [PATCH, v5 2/2] cgroups: introduce timer slack subsystem
Posted by Kirill A. Shutsemov on Mon, 07 Feb 2011 09:46:02 GMT

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

Provides a way of tasks grouping by timer slack value. Introduces per
cgroup max and min timer slack value. When a task attaches to a cgroup,
its timer slack value adjusts (if needed) to fit min-max range.

It also provides a way to set timer slack value for all tasks in the
cgroup at once.

This functionality is useful in mobile devices where certain background
apps are attached to a cgroup and minimum wakeups are desired.

Idea-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
---
 include/linux/cgroup_subsys.h | 6 +
 include/linux/init_task.h | 4 +-
 init/Kconfig | 10 ++
 kernel/Makefile | 1 +
 kernel/cgroup_timer_slack.c | 281 ++
 kernel/sys.c | 19 ++-
 6 files changed, 314 insertions(+), 7 deletions(-)
 create mode 100644 kernel/cgroup_timer_slack.c

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/include/linux/init_task.h b/include/linux/init_task.h
index caa151f..48eca8f 100644
--- a/include/linux/init_task.h
+++ b/include/linux/init_task.h
@@ -124,6 +124,8 @@ extern struct cred init_cred;
 # define INIT_PERF_EVENTS(tsk)
 #endif

+#define TIMER_SLACK_NS_DEFAULT 50000

```
+
 /*
  *  INIT_TASK is used to set up the first task table, touch at
  * your own risk!. Base=0, limit=0x1fffff (=2MB)
@@ -177,7 +179,7 @@ extern struct cred init_cred;
  .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers),  \
  .fs_excl = ATOMIC_INIT(0),    \
  .pi_lock = __RAW_SPIN_LOCK_UNLOCKED(tsk.pi_lock), \
- .timer_slack_ns = 50000, /* 50 usec default slack */  \
+ .timer_slack_ns = TIMER_SLACK_NS_DEFAULT,   \
  .pids = {       \
   [PIDTYPE_PID]  = INIT_PID_LINK(PIDTYPE_PID),  \
   [PIDTYPE_PGID] = INIT_PID_LINK(PIDTYPE_PGID),  \
diff --git a/init/Kconfig b/init/Kconfig
index be788c0..6cf465f 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 subsystem"
+ help
+   Provides a way of tasks grouping by timer slack value.
+   Introduces per cgroup timer slack value which will override
+   the default timer slack value once a task is attached to a
+   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 353d3fe..0b60239 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..609e1f1
--- /dev/null
+++ b/kernel/cgroup_timer_slack.c
@@ -0,0 +1,281 @@
+/*
+ * 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,
+};
+
+extern int (*timer_slack_check)(struct task_struct *task,
+  unsigned long slack_ns);
+
+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 task_struct *task,
+  unsigned long slack_ns)
+{
+ struct cgroup_subsys_state *css;
+ struct timer_slack_cgroup *tslack_cgroup;
+
+ /* XXX: lockdep false positive? */
+ rcu_read_lock();
+ css = task_subsys_state(task, 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 -EPERM;
+ return 0;
+}
+
+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 (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;
+ else if (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;
+  else if (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,
+#ifdef CONFIG_CGROUP_TIMER_SLACK
+ .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;
+
+ if (timer_slack_check)
+  return -EBUSY;
+ timer_slack_check = cgroup_timer_slack_check;
+
+ err = cgroup_load_subsys(&timer_slack_subsys);
+ if (err)
+  timer_slack_check = NULL;
+ return err;
+}
+
+static void __exit exit_cgroup_timer_slack(void)
+{
+ BUG_ON(timer_slack_check != cgroup_timer_slack_check);
+ timer_slack_check = NULL;
+ cgroup_unload_subsys(&timer_slack_subsys);
+}
+
+module_init(init_cgroup_timer_slack);
+module_exit(exit_cgroup_timer_slack);
+MODULE_LICENSE("GPL");
diff --git a/kernel/sys.c b/kernel/sys.c
index 18da702..3bc4205 100644
--- a/kernel/sys.c
```

```
+++ b/kernel/sys.c
@@ -118,6 +118,10 @@ EXPORT_SYMBOL(cad_pid);

 void (*pm_power_off_prepare)(void);

+int (*timer_slack_check)(struct task_struct *task, unsigned long slack_ns) =
+ NULL;
+EXPORT_SYMBOL_GPL(timer_slack_check);
+
 /*
  * set the priority of a task
  * - the caller must hold the RCU read lock
@@ -1694,12 +1698,15 @@ SYSCALL_DEFINE5(prctl, int, option, unsigned long, arg2,
unsigned long, arg3,
    error = current->timer_slack_ns;
    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;
+   if (arg2 <= 0) {
+    me->timer_slack_ns = me->default_timer_slack_ns;
+    break;
+   }
+
+   error = timer_slack_check ?
+    timer_slack_check(me, arg2) : 0;
+   if (!error)
+    me->timer_slack_ns = arg2;
    break;
   case PR_MCE_KILL:
    if (arg4 | arg5)
--
1.7.4
```

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

---

Subject: Re: [PATCH, v5 1/2] cgroups: export cgroup_iter_{start, next, end}
Posted by Balbir Singh on Mon, 07 Feb 2011 09:57:13 GMT
View Forum Message <> Reply to Message

* Kirill A. Shutsemov <kirill@shutemov.name> [2011-02-07 11:46:01]:

> From: Kirill A. Shutemov <kirill@shutemov.name>
>
> Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
> Acked-by: Paul Menage <menage@google.com>

Acked-by: Balbir Singh <balbir@linux.vnet.ibm.com>

--
Three Cheers,
Balbir

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

---

Subject: Re: [PATCH, v5 2/2] cgroups: introduce timer slack subsystem
Posted by Balbir Singh on Mon, 07 Feb 2011 10:06:24 GMT
View Forum Message <> Reply to Message

* Kirill A. Shutsemov <kirill@shutemov.name> [2011-02-07 11:46:02]:

> From: Kirill A. Shutemov <kirill@shutemov.name>
>
> Provides a way of tasks grouping by timer slack value. Introduces per
> cgroup max and min timer slack value. When a task attaches to a cgroup,
> its timer slack value adjusts (if needed) to fit min-max range.
>
> It also provides a way to set timer slack value for all tasks in the
> cgroup at once.
>
> This functionality is useful in mobile devices where certain background
> apps are attached to a cgroup and minimum wakeups are desired.
>
> Idea-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
> Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
> ---
> include/linux/cgroup_subsys.h | 6 +
> include/linux/init_task.h | 4 +-
> init/Kconfig | 10 ++
> kernel/Makefile | 1 +
> kernel/cgroup_timer_slack.c | 281 ++
> kernel/sys.c | 19 ++-
> 6 files changed, 314 insertions(+), 7 deletions(-)

> create mode 100644 kernel/cgroup_timer_slack.c
>
> 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/include/linux/init_task.h b/include/linux/init_task.h
> index caa151f..48eca8f 100644
> --- a/include/linux/init_task.h
> +++ b/include/linux/init_task.h
> @@ -124,6 +124,8 @@ extern struct cred init_cred;
> # define INIT_PERF_EVENTS(tsk)
> #endif
>
> +#define TIMER_SLACK_NS_DEFAULT 50000
> +
> /*
>   * INIT_TASK is used to set up the first task table, touch at
>   * your own risk!. Base=0, limit=0x1fffff (=2MB)
> @@ -177,7 +179,7 @@ extern struct cred init_cred;
>   .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers), \
>   .fs_excl = ATOMIC_INIT(0),   \
>   .pi_lock = __RAW_SPIN_LOCK_UNLOCKED(tsk.pi_lock), \
> - .timer_slack_ns = 50000, /* 50 usec default slack */ \
> + .timer_slack_ns = TIMER_SLACK_NS_DEFAULT,   \
>   .pids = {   \
>   [PIDTYPE_PID] = INIT_PID_LINK(PIDTYPE_PID), \
>   [PIDTYPE_PGID] = INIT_PID_LINK(PIDTYPE_PGID), \
> diff --git a/init/Kconfig b/init/Kconfig
> index be788c0..6cf465f 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 subsystem"

> + help
> +   Provides a way of tasks grouping by timer slack value.
> +   Introduces per cgroup timer slack value which will override
> +   the default timer slack value once a task is attached to a
> +   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 353d3fe..0b60239 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..609e1f1
> --- /dev/null
> +++ b/kernel/cgroup_timer_slack.c
> @@ -0,0 +1,281 @@
> +/*
> + * 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,
> +};
> +
> +extern int (*timer_slack_check)(struct task_struct *task,
> +  unsigned long slack_ns);
> +
> +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 task_struct *task,
> +  unsigned long slack_ns)
> +{
> + struct cgroup_subsys_state *css;
> + struct timer_slack_cgroup *tslack_cgroup;
> +
> + /* XXX: lockdep false positive? */
> + rcu_read_lock();
> + css = task_subsys_state(task, timer_slack_subsys.subsys_id);
> + tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
```

Shouldn't the rcu_read_unlock() be after the check of is_timer_slack_allowed()?

```
> + rcu_read_unlock();
> +
> + if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
> +  return -EPERM;
> + return 0;
> +}
> +
> +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);
```

OK, I've got questions, are the following possible

1. Assign a task to a cgroup based on timer slack, it is accepted
based on min and max values
2. We update the slack

Can't this be bad for some tasks, is there a way for a task to prevent
the max slack from being updated?

2. Can there be an overlap in the ranges?

--
Three Cheers,
Balbir

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

Subject: Re: [PATCH, v5 2/2] cgroups: introduce timer slack subsystem
Posted by Kirill A. Shutsemov on Mon, 07 Feb 2011 10:57:30 GMT
View Forum Message <> Reply to Message

On Mon, Feb 07, 2011 at 03:36:24PM +0530, Balbir Singh wrote:
> * Kirill A. Shutsemov <kirill@shutemov.name> [2011-02-07 11:46:02]:
>
> > From: Kirill A. Shutemov <kirill@shutemov.name>
> >
> > Provides a way of tasks grouping by timer slack value. Introduces per
> > cgroup max and min timer slack value. When a task attaches to a cgroup,
> > its timer slack value adjusts (if needed) to fit min-max range.
> >
> > It also provides a way to set timer slack value for all tasks in the
> > cgroup at once.
> >
> > This functionality is useful in mobile devices where certain background
> > apps are attached to a cgroup and minimum wakeups are desired.
> >
> > Idea-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
> > Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
> > ---
> > include/linux/cgroup_subsys.h | 6 +
> > include/linux/init_task.h | 4 +-
> > init/Kconfig | 10 ++
> > kernel/Makefile | 1 +
> > kernel/cgroup_timer_slack.c | 281 ++
> > kernel/sys.c | 19 ++-
> > 6 files changed, 314 insertions(+), 7 deletions(-)
> > create mode 100644 kernel/cgroup_timer_slack.c
> >
> > 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/include/linux/init_task.h b/include/linux/init_task.h
> > index caa151f..48eca8f 100644
> > --- a/include/linux/init_task.h
> > +++ b/include/linux/init_task.h

> > @@ -124,6 +124,8 @@ extern struct cred init_cred;
> > # define INIT_PERF_EVENTS(tsk)
> > #endif
> >
> > +#define TIMER_SLACK_NS_DEFAULT 50000
> > +
> > /*
> >   * INIT_TASK is used to set up the first task table, touch at
> >   * your own risk!. Base=0, limit=0x1fffff (=2MB)
> > @@ -177,7 +179,7 @@ extern struct cred init_cred;
> >   .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers), \
> >   .fs_excl = ATOMIC_INIT(0),   \
> >   .pi_lock = __RAW_SPIN_LOCK_UNLOCKED(tsk.pi_lock), \
> > - .timer_slack_ns = 50000, /* 50 usec default slack */ \
> > + .timer_slack_ns = TIMER_SLACK_NS_DEFAULT,   \
> >   .pids = {   \
> >   [PIDTYPE_PID] = INIT_PID_LINK(PIDTYPE_PID), \
> >   [PIDTYPE_PGID] = INIT_PID_LINK(PIDTYPE_PGID), \
> > diff --git a/init/Kconfig b/init/Kconfig
> > index be788c0..6cf465f 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 subsystem"
> > + help
> > +   Provides a way of tasks grouping by timer slack value.
> > +   Introduces per cgroup timer slack value which will override
> > +   the default timer slack value once a task is attached to a
> > +   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 353d3fe..0b60239 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..609e1f1
> > --- /dev/null
> > +++ b/kernel/cgroup_timer_slack.c
> > @@ -0,0 +1,281 @@
> > +/*
> > + * 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,
> > +};
> > +
> > +extern int (*timer_slack_check)(struct task_struct *task,
> > + unsigned long slack_ns);
> > +
> > +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 task_struct *task,
> > + unsigned long slack_ns)
> > +{
> > + struct cgroup_subsys_state *css;
> > + struct timer_slack_cgroup *tslack_cgroup;
> > +
> > + /* XXX: lockdep false positive? */
> > + rcu_read_lock();
> > + css = task_subsys_state(task, timer_slack_subsys.subsys_id);
> > + tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
>
> Shouldn't the rcu_read_unlock() be after the check of is_timer_slack_allowed()?

I'm not sure if this rcu_read_lock() is needed. See freezer_fork().

> > + rcu_read_unlock();
> > +
> > + if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
> > + return -EPERM;
> > + return 0;
> > +}
> > +
> > +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);
>
> OK, I've got questions, are the following possible
>
> 1. Assign a task to a cgroup based on timer slack, it is accepted
> based on min and max values

Hm.. Currently, there is no such functionality. Do we need it? What's use
case?

> 2. We update the slack
>
> Can't this be bad for some tasks, is there a way for a task to prevent
> the max slack from being updated?

If a non-RT task can't tolerate not being on time it's totally broken.
RT-tasks are not affected by timer slack.

And we believe that user knows what he does.

> 2. Can there be an overlap in the ranges?

What overlap do you mean?
There is only one rule: parent's range should be wider then child's.

--
 Kirill A. Shutemov

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

---

Subject: Re: [PATCH, v5 2/2] cgroups: introduce timer slack subsystem
Posted by Balbir Singh on Mon, 07 Feb 2011 12:14:30 GMT
View Forum Message <> Reply to Message

* Kirill A. Shutemov <kirill@shutemov.name> [2011-02-07 12:57:30]:

> On Mon, Feb 07, 2011 at 03:36:24PM +0530, Balbir Singh wrote:
> > * Kirill A. Shutsemov <kirill@shutemov.name> [2011-02-07 11:46:02]:
> >
> > > From: Kirill A. Shutemov <kirill@shutemov.name>
> > >
> > > Provides a way of tasks grouping by timer slack value. Introduces per
> > > cgroup max and min timer slack value. When a task attaches to a cgroup,
> > > its timer slack value adjusts (if needed) to fit min-max range.
> > >
> > > It also provides a way to set timer slack value for all tasks in the
> > > cgroup at once.
> > >
> > > This functionality is useful in mobile devices where certain background
> > > apps are attached to a cgroup and minimum wakeups are desired.
> > >
> > > Idea-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
> > > Signed-off-by: Kirill A. Shutemov <kirill@shutemov.name>
> > > ---
> > > include/linux/cgroup_subsys.h | 6 +
> > > include/linux/init_task.h | 4 +-
> > > init/Kconfig | 10 ++
> > > kernel/Makefile | 1 +
> > > kernel/cgroup_timer_slack.c | 281 ++
> > > kernel/sys.c | 19 ++-
> > > 6 files changed, 314 insertions(+), 7 deletions(-)
> > > create mode 100644 kernel/cgroup_timer_slack.c
> > >
> > > 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/include/linux/init_task.h b/include/linux/init_task.h
> > > index caa151f..48eca8f 100644
> > > --- a/include/linux/init_task.h
> > > +++ b/include/linux/init_task.h
> > > @@ -124,6 +124,8 @@ extern struct cred init_cred;
> > > # define INIT_PERF_EVENTS(tsk)
> > > #endif

```
> > >
> > > +#define TIMER_SLACK_NS_DEFAULT 50000
> > > +
> > >  /*
> > >   *  INIT_TASK is used to set up the first task table, touch at
> > >   * your own risk!. Base=0, limit=0x1fffff (=2MB)
> > > @@ -177,7 +179,7 @@ extern struct cred init_cred;
> > >   .cpu_timers = INIT_CPU_TIMERS(tsk.cpu_timers),  \
> > >   .fs_excl = ATOMIC_INIT(0),    \
> > >   .pi_lock = __RAW_SPIN_LOCK_UNLOCKED(tsk.pi_lock), \
> > > - .timer_slack_ns = 50000, /* 50 usec default slack */  \
> > > + .timer_slack_ns = TIMER_SLACK_NS_DEFAULT,   \
> > >   .pids = {       \
> > >    [PIDTYPE_PID]  = INIT_PID_LINK(PIDTYPE_PID),  \
> > >    [PIDTYPE_PGID] = INIT_PID_LINK(PIDTYPE_PGID),  \
> > > diff --git a/init/Kconfig b/init/Kconfig
> > > index be788c0..6cf465f 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 subsystem"
> > > + help
> > > +   Provides a way of tasks grouping by timer slack value.
> > > +   Introduces per cgroup timer slack value which will override
> > > +   the default timer slack value once a task is attached to a
> > > +   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 353d3fe..0b60239 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..609e1f1
> > > --- /dev/null
> > > +++ b/kernel/cgroup_timer_slack.c
> > > @@ -0,0 +1,281 @@
> > > +/*
> > > + * 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,
> > > +};
> > > +
> > > +extern int (*timer_slack_check)(struct task_struct *task,
> > > +  unsigned long slack_ns);
> > > +
> > > +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 task_struct *task,
> > > + unsigned long slack_ns)
> > > +{
> > > + struct cgroup_subsys_state *css;
> > > + struct timer_slack_cgroup *tslack_cgroup;
> > > +
> > > + /* XXX: lockdep false positive? */
> > > + rcu_read_lock();
> > > + css = task_subsys_state(task, timer_slack_subsys.subsys_id);
> > > + tslack_cgroup = container_of(css, struct timer_slack_cgroup, css);
> >
> > Shouldn't the rcu_read_unlock() be after the check of is_timer_slack_allowed()?
>
> I'm not sure if this rcu_read_lock() is needed. See freezer_fork().
>
> > > + rcu_read_unlock();
> > > +
> > > + if (!is_timer_slack_allowed(tslack_cgroup, slack_ns))
> > > + return -EPERM;
> > > + return 0;
> > > +}
> > > +
> > > +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;

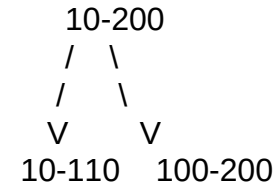
```
> > > + } else {
> > > +  tslack_cgroup->min_slack_ns = 0UL;
> > > +  tslack_cgroup->max_slack_ns = ULONG_MAX;
> > > + }
> > > +
> > > + return &tslack_cgroup->css;
> > > +}
> > > +
> > > +static void tslack_cgroup_destroy(struct cgroup_subsys *subsys,
> > > +  struct cgroup *cgroup)
> > > +{
> > > + 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);
> >
> > OK, I've got questions, are the following possible
> >
> > 1. Assign a task to a cgroup based on timer slack, it is accepted
> > based on min and max values
>
> Hm.. Currently, there is no such functionality. Do we need it? What's use
> case?
>
> > 2. We update the slack
> >
> > Can't this be bad for some tasks, is there a way for a task to prevent
> > the max slack from being updated?
>
> If a non-RT task can't tolerate not being on time it's totally broken.

Yes, I agree, except that the slack might be overwritten after the
application set an acceptable slack.

> RT-tasks are not affected by timer slack.
>
> And we believe that user knows what he does.
>
> > 2. Can there be an overlap in the ranges?
>
> What overlap do you mean?
> There is only one rule: parent's range should be wider then child's.

Between the children, can we have something like



--
Three Cheers,
Balbir

_____
Containers mailing list

Subject: Re: [PATCH, v5 2/2] cgroups: introduce timer slack subsystem
Posted by Kirill A. Shutsemov on Mon, 07 Feb 2011 12:26:28 GMT
View Forum Message <> Reply to Message

On Mon, Feb 07, 2011 at 05:44:30PM +0530, Balbir Singh wrote:
> * Kirill A. Shutemov <kirill@shutemov.name> [2011-02-07 12:57:30]:
>
> > On Mon, Feb 07, 2011 at 03:36:24PM +0530, Balbir Singh wrote:
> > > 2. Can there be an overlap in the ranges?
> >
> > What overlap do you mean?
> > There is only one rule: parent's range should be wider then child's.
>
> Between the children, can we have something like
>
>   10-200
>   /   \
>   /   \
>   V   V
>   10-110   100-200

Yes, sure.

--
Kirill A. Shutemov

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