Subject: Re: [PATCH 1/1, v9] cgroup/freezer: add per freezer duty ratio control
Posted by rjw on Wed, 16 Feb 2011 00:00:15 GMT
View Forum Message <> Reply to Message

On Tuesday, February 15, 2011, jacob.jun.pan@linux.intel.com wrote:
> From: Jacob Pan <jacob.jun.pan@linux.intel.com>
>
> Freezer subsystem is used to manage batch jobs which can start
> stop at the same time. However, sometime it is desirable to let
> the kernel manage the freezer state automatically with a given
> duty ratio.
> For example, if we want to reduce the time that backgroup apps
> are allowed to run we can put them into a freezer subsystem and
> set the kernel to turn them THAWED/FROZEN at given period and
> percentage of frozen time.
>
> This patch introduces two file nodes under cgroup
> freezer.period_second
> freezer.frozen_time_percentage
>
> Usage example: set period to be 5 seconds and frozen duty ratio 90%
> [root@localhost aoa]# echo 90 > freezer.frozen_time_percentage
> [root@localhost aoa]# echo 5 > freezer.period_second
>
> Signed-off-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
> ---
>  Documentation/cgroups/freezer-subsystem.txt |  33 ++++++
>  kernel/cgroup_freezer.c                     | 155 ++++++++++++++++++++++++++-
>  2 files changed, 185 insertions(+), 3 deletions(-)
>
> diff --git a/Documentation/cgroups/freezer-subsystem.txt
b/Documentation/cgroups/freezer-subsystem.txt
> index 41f37fe..df0f998 100644
> --- a/Documentation/cgroups/freezer-subsystem.txt
> +++ b/Documentation/cgroups/freezer-subsystem.txt
> @@ -100,3 +100,36 @@ things happens:
>    and returns EINVAL)
>  3) The tasks that blocked the cgroup from entering the "FROZEN"
>    state disappear from the cgroup's set of tasks.
> +
> +In embedded systems, it is desirable to manage group of applications
> +for power saving. E.g. tasks that are not in the foreground may be
> +frozen and unfrozen periodically to save power without affecting user
> +experience. In this case, user/management software can attach tasks
> +into freezer cgroup then specify a period and percentage of time the
> +tasks within the cgroup shall be frozen.
> +
> +100% frozen will result in the same behavior as setting freezer.state

> +to FROZEN. Similarly, 0% is equivalent to setting freezer.state to
> +THAWED.
> +The granularity of the period is second, due to the fact that small
> +period may result in greater overhead and causes stability issue since
> +common workqueue is used here to perform the management.
> +
> +Zero value of period_second is not allowed, the default period is 1
> +second.
> +
> +Usage example:
> +Assuming freezer cgroup is already mounted, application being managed
> +are included the "tasks" file node of the given freezer cgroup.
> +To make the tasks frozen at 90% of the time every 5 seconds, do:
> +
> +[root@localhost ]# echo 90 > freezer.frozen_time_percentage
> +[root@localhost ]# echo 5 > freezer.period_second

Well, are you trying to work around a problem with some applications that
cause too many wakeups to occur?

Why do you need the kernel to do that for you on a per-cgroup basis?
What about simply sending SIGSTOP/SIGCONT periodically from user space to the
applications in question?

Moreover, what about having a user space process that will freeze/unfreeze the
entire cgroup as needed?

> +After that, the application in this freezer cgroup will only be
> +allowed to run at the following pattern.
> +
> +
> +
> +

How are you going to handle dependencies between the "periodically frozen"
applications and other applications that may need them to do some work?

> diff --git a/kernel/cgroup_freezer.c b/kernel/cgroup_freezer.c
> index e7bebb7..32beacb 100644
> --- a/kernel/cgroup_freezer.c
> +++ b/kernel/cgroup_freezer.c
> @@ -28,12 +28,31 @@ enum freezer_state {
>  CGROUP_FROZEN,
>  };
>
> +enum duty_ratio_params {
> + FREEZER_DUTY_RATIO = 0,

> + FREEZER_PERIOD,
> +};
> +
> +struct freezer_duty {
> + u32 ratio; /* percentage of time frozen */
> + u32 period_pct_ms; /* one percent of the period in miliseconds */

What period?

> + u32 enabled;
> +};
> +
>  struct freezer {
>   struct cgroup_subsys_state css;
>   enum freezer_state state;
> + struct freezer_duty duty;
> + struct delayed_work freezer_work; /* work to duty-cycle a cgroup */
>   spinlock_t lock; /* protects _writes_ to state */
>  };
>
> +static int freezer_change_state(struct cgroup *cgroup,
> +    enum freezer_state goal_state);
> +static void unfreeze_cgroup(struct cgroup *cgroup, struct freezer *freezer);
> +static int try_to_freeze_cgroup(struct cgroup *cgroup, struct freezer *freezer);
> +static void update_if_frozen(struct cgroup *cgroup, struct freezer *freezer);
> +
>  static inline struct freezer *cgroup_freezer(
>    struct cgroup *cgroup)
>  {
> @@ -63,6 +82,35 @@ int cgroup_freezing_or_frozen(struct task_struct *task)
>   return result;
>  }
>
> +static void freezer_work_fn(struct work_struct *work)
> +{
> + struct freezer *freezer;
> + unsigned long delay_jiffies = 0;
> +
> + freezer = container_of(work, struct freezer, freezer_work.work);
> + spin_lock_irq(&freezer->lock);

So you switch interrupts off every once a while simply to freeze/unfreeze
your cgroup.  That's going to affect the scheduler quite a bit, isn't it?

> + /* toggle between THAWED and FROZEN state. */
> + update_if_frozen(freezer->css.cgroup, freezer);

What about using a local variable cgroup = freezer->css.cgroup ?

> + switch (freezer->state) {
> + case CGROUP_FROZEN:
> + case CGROUP_FREEZING:
> +  delay_jiffies = msecs_to_jiffies((100 - freezer->duty.ratio) *
> +      freezer->duty.period_pct_ms);
> +  unfreeze_cgroup(freezer->css.cgroup, freezer);
> +  break;
> + case CGROUP_THAWED:
> +  delay_jiffies = msecs_to_jiffies(freezer->duty.ratio *
> +      freezer->duty.period_pct_ms);
> +  try_to_freeze_cgroup(freezer->css.cgroup, freezer);
> +  break;
> + default:
> +  BUG();

Don't crash the kernel if you don't have to.

> + }
> +
> + schedule_delayed_work(&freezer->freezer_work, delay_jiffies);

What if this work happens to run during system suspend/resume?

> + spin_unlock_irq(&freezer->lock);
> +}
> +
>  /*
>   * cgroups_write_string() limits the size of freezer state strings to
>   * CGROUP_LOCAL_BUFFER_SIZE
> @@ -144,13 +192,19 @@ static struct cgroup_subsys_state *freezer_create(struct
cgroup_subsys *ss,
>
>   spin_lock_init(&freezer->lock);
>   freezer->state = CGROUP_THAWED;
> + freezer->duty.period_pct_ms = 10; /* default to 1 second period */

Why this number?

>   return &freezer->css;
>  }
>
>  static void freezer_destroy(struct cgroup_subsys *ss,
>        struct cgroup *cgroup)
>  {
> - kfree(cgroup_freezer(cgroup));
> + struct freezer *freezer;
> +

> + freezer = cgroup_freezer(cgroup);
> + if (freezer->duty.enabled)
> +  cancel_delayed_work_sync(&freezer->freezer_work);
> + kfree(freezer);
>  }
>
>  /*
> @@ -282,6 +336,16 @@ static int freezer_read(struct cgroup *cgroup, struct cftype *cft,
>   return 0;
>  }
>
> +static u64 freezer_read_duty_ratio(struct cgroup *cgroup, struct cftype *cft)
> +{
> + return cgroup_freezer(cgroup)->duty.ratio;
> +}
> +
> +static u64 freezer_read_period(struct cgroup *cgroup, struct cftype *cft)
> +{
> + return cgroup_freezer(cgroup)->duty.period_pct_ms / 10;
> +}
> +
>  static int try_to_freeze_cgroup(struct cgroup *cgroup, struct freezer *freezer)
>  {
>   struct cgroup_iter it;
> @@ -347,20 +411,56 @@ out:
>   return retval;
>  }
>

It would be _nice_ to add a kerneldoc comment explaining the purpose of the
new function, right?

> +static void freezer_check_duty_cycling(struct freezer *freezer, int set_state)
> +{
> + if (freezer->duty.enabled) {
> +  if (freezer->duty.ratio && freezer->duty.ratio < 100)
> +   return;
> +  pr_debug("stopping duty ratio mode\n");
> +  cancel_delayed_work_sync(&freezer->freezer_work);
> +  freezer->duty.enabled = 0;
> +  if (set_state) {
> +   update_if_frozen(freezer->css.cgroup, freezer);
> +   if (freezer->duty.ratio == 100)
> +    try_to_freeze_cgroup(freezer->css.cgroup,
> +      freezer);
> +   else if (freezer->duty.ratio == 0)
> +    unfreeze_cgroup(freezer->css.cgroup, freezer);
> +   else

> +    BUG();

Again, don't crash the kernel if you don't have to.

> +  }
> + } else if (freezer->duty.ratio < 100 && freezer->duty.ratio) {
> +  pr_debug("starting duty ratio mode\n");
> +  INIT_DELAYED_WORK(&freezer->freezer_work, freezer_work_fn);
> +  freezer->duty.enabled = 1;
> +  schedule_delayed_work(&freezer->freezer_work, 0);
> + }
> +}
> +
>  static int freezer_write(struct cgroup *cgroup,
>       struct cftype *cft,
>       const char *buffer)
>  {
>   int retval;
>   enum freezer_state goal_state;
> + struct freezer *freezer;
>
> - if (strcmp(buffer, freezer_state_strs[CGROUP_THAWED]) == 0)
> + freezer = cgroup_freezer(cgroup);
> + if (strcmp(buffer, freezer_state_strs[CGROUP_THAWED]) == 0) {
>    goal_state = CGROUP_THAWED;
> - else if (strcmp(buffer, freezer_state_strs[CGROUP_FROZEN]) == 0)
> +  freezer->duty.ratio = 0;
> + } else if (strcmp(buffer, freezer_state_strs[CGROUP_FROZEN]) == 0) {
> +  freezer->duty.ratio = 100;
>    goal_state = CGROUP_FROZEN;
> + }
>   else
>    return -EINVAL;
>
> + /* we should stop duty ratio toggling if user wants to
> +  * force change to a valid state.
> +  */
> + freezer_check_duty_cycling(freezer, 0);

Doesn't that need to be under the lock?

> +
>   if (!cgroup_lock_live_group(cgroup))
>    return -ENODEV;
>   retval = freezer_change_state(cgroup, goal_state);
> @@ -368,12 +468,61 @@ static int freezer_write(struct cgroup *cgroup,
>   return retval;
>  }

>

A kerneldoc comment would also be nice to have here.

> +static int freezer_write_param(struct cgroup *cgroup, struct cftype *cft,
> +  u64 val)
> +{
> + struct freezer *freezer;
> + int ret = 0;
> +
> + freezer = cgroup_freezer(cgroup);
> + if (!cgroup_lock_live_group(cgroup))
> +  return -ENODEV;
> +
> + spin_lock_irq(&freezer->lock);
> + switch (cft->private) {
> + case FREEZER_DUTY_RATIO:
> +  if (val > 100) {
> +   ret = -EINVAL;
> +   goto exit;
> +  }
> +  freezer->duty.ratio = val;
> +  break;
> + case FREEZER_PERIOD:
> +  if (!val) {
> +   ret = -EINVAL;
> +   goto exit;
> +  }
> +  freezer->duty.period_pct_ms = val * 10;
> +  break;
> + default:
> +  BUG();

Rule: Use BUG() only if the consequences of not doing so would be _worse_
than a kernel crash.

> + }
> + spin_unlock_irq(&freezer->lock);
> + freezer_check_duty_cycling(freezer, 1);
> +exit:
> + cgroup_unlock();
> + return ret;
> +}
> +
>  static struct cftype files[] = {
>   {
>    .name = "state",
>    .read_seq_string = freezer_read,

>  .write_string = freezer_write,
>  },
> + {
> +  .name = "frozen_time_percentage",
> +  .private = FREEZER_DUTY_RATIO,
> +  .read_u64 = freezer_read_duty_ratio,
> +  .write_u64 = freezer_write_param,
> + },
> + {
> +  .name = "period_second",
> +  .private = FREEZER_PERIOD,
> +  .read_u64 = freezer_read_period,
> +  .write_u64 = freezer_write_param,
> + },
> +
>  };
>
>  static int freezer_populate(struct cgroup_subsys *ss, struct cgroup *cgroup)

Well, quite frankly, I'm not going to take this patch unless it gets an ACK
from the scheduler people (which I'm guessing is not going to happen before
hell freezes over).

IOW, please find a better way to address the issue at hand.

Thanks,
Rafael

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

---

Subject: Re: [PATCH 1/1, v9] cgroup/freezer: add per freezer duty ratio control
Posted by jacob.jun.pan on Wed, 16 Feb 2011 00:38:12 GMT
View Forum Message <> Reply to Message

On Wed, 16 Feb 2011 01:00:15 +0100
"Rafael J. Wysocki" <rjw@sisk.pl> wrote:

> On Tuesday, February 15, 2011, jacob.jun.pan@linux.intel.com wrote:
> > From: Jacob Pan <jacob.jun.pan@linux.intel.com>
> >
> > Freezer subsystem is used to manage batch jobs which can start
> > stop at the same time. However, sometime it is desirable to let
> > the kernel manage the freezer state automatically with a given
> > duty ratio.
> > For example, if we want to reduce the time that backgroup apps

> > are allowed to run we can put them into a freezer subsystem and
> > set the kernel to turn them THAWED/FROZEN at given period and
> > percentage of frozen time.
> >
> > This patch introduces two file nodes under cgroup
> > freezer.period_second
> > freezer.frozen_time_percentage
> >
> > Usage example: set period to be 5 seconds and frozen duty ratio 90%
> > [root@localhost aoa]# echo 90 > freezer.frozen_time_percentage
> > [root@localhost aoa]# echo 5 > freezer.period_second
> >
> > Signed-off-by: Jacob Pan <jacob.jun.pan@linux.intel.com>
> > ---
> >  Documentation/cgroups/freezer-subsystem.txt |   33 ++++++
> >  kernel/cgroup_freezer.c                     |  155
> > ++++++++++++++++++++++++++++- 2 files changed, 185 insertions(+), 3
> > deletions(-)
> >
> > diff --git a/Documentation/cgroups/freezer-subsystem.txt
> > b/Documentation/cgroups/freezer-subsystem.txt index
> > 41f37fe..df0f998 100644 ---
> > a/Documentation/cgroups/freezer-subsystem.txt +++
> > b/Documentation/cgroups/freezer-subsystem.txt @@ -100,3 +100,36 @@
> > things happens: and returns EINVAL)
> >   3) The tasks that blocked the cgroup from entering the
> > "FROZEN" state disappear from the cgroup's set of tasks.
> > +
> > +In embedded systems, it is desirable to manage group of
> > applications +for power saving. E.g. tasks that are not in the
> > foreground may be +frozen and unfrozen periodically to save power
> > without affecting user +experience. In this case, user/management
> > software can attach tasks +into freezer cgroup then specify a
> > period and percentage of time the +tasks within the cgroup shall be
> > frozen. +
> > +100% frozen will result in the same behavior as setting
> > freezer.state +to FROZEN. Similarly, 0% is equivalent to setting
> > freezer.state to +THAWED.
> > +The granularity of the period is second, due to the fact that small
> > +period may result in greater overhead and causes stability issue
> > since +common workqueue is used here to perform the management.
> > +
> > +Zero value of period_second is not allowed, the default period is 1
> > +second.
> > +
> > +Usage example:
> > +Assuming freezer cgroup is already mounted, application being
> > managed +are included the "tasks" file node of the given freezer

> > cgroup. +To make the tasks frozen at 90% of the time every 5
> > seconds, do: +
> > +[root@localhost ]# echo 90 > freezer.frozen_time_percentage
> > +[root@localhost ]# echo 5 > freezer.period_second
>
> Well, are you trying to work around a problem with some applications
> that cause too many wakeups to occur?
>
not exactly. I am trying to use this feature to manage power thermal
aspect as well.
> Why do you need the kernel to do that for you on a per-cgroup basis?
> What about simply sending SIGSTOP/SIGCONT periodically from user
> space to the applications in question?
>
I thought Documentation/cgroups/freezer-subsystem.tx explains the
reason why "SIGSTOP and SIGCONT are not always sufficient for stopping
and resuming tasks in userspace".
> Moreover, what about having a user space process that will
> freeze/unfreeze the entire cgroup as needed?
>
Yes, it is doable in the userspace, but managing in userspace has
greater overhead.

> > +After that, the application in this freezer cgroup will only be
> > +allowed to run at the following pattern.
> > +
> > +    __                  __                  __
> > +    |  |<-- 90% frozen -->|  |                |  |
> > +____|  |__________________|  |________________|  |_____
> > +
> > +    |<---- 5 seconds ---->|
>
> How are you going to handle dependencies between the "periodically
> frozen" applications and other applications that may need them to do
> some work?
>
I don't have an answer, but first of all when we launch an app, it
makes sure all its tasks (children) are in the same cgroup and the
common services are allowed to run all the time, this should reduce the
dependencies. Secondly, my goal is to keep the apps alive such that
certain power/thermal targets can be met, under some cases
such as when the screen is off or an app is not in the foreground.
Sometimes, the only goal is to keep them alive, perhaps we need some
tuning to find that frozen time percentage limit.

IMHO, w/o this patch, the dependency problem still exists, true?

> > diff --git a/kernel/cgroup_freezer.c b/kernel/cgroup_freezer.c
> > index e7bebb7..32beacb 100644

> > --- a/kernel/cgroup_freezer.c
> > +++ b/kernel/cgroup_freezer.c
> > @@ -28,12 +28,31 @@ enum freezer_state {
> >  CGROUP_FROZEN,
> >  };
> >
> > +enum duty_ratio_params {
> > + FREEZER_DUTY_RATIO = 0,
> > + FREEZER_PERIOD,
> > +};
> > +
> > +struct freezer_duty {
> > + u32 ratio; /* percentage of time frozen */
> > + u32 period_pct_ms; /* one percent of the period in
> > miliseconds */
>
> What period?
>
> > + u32 enabled;
> > +};
> > +
> >  struct freezer {
> >  struct cgroup_subsys_state css;
> >  enum freezer_state state;
> > + struct freezer_duty duty;
> > + struct delayed_work freezer_work; /* work to duty-cycle a
> > cgroup */ spinlock_t lock; /* protects _writes_ to state */
> >  };
> >
> > +static int freezer_change_state(struct cgroup *cgroup,
> > +  enum freezer_state goal_state);
> > +static void unfreeze_cgroup(struct cgroup *cgroup, struct freezer
> > *freezer); +static int try_to_freeze_cgroup(struct cgroup *cgroup,
> > struct freezer *freezer); +static void update_if_frozen(struct
> > cgroup *cgroup, struct freezer *freezer); +
> >  static inline struct freezer *cgroup_freezer(
> >  struct cgroup *cgroup)
> >  {
> > @@ -63,6 +82,35 @@ int cgroup_freezing_or_frozen(struct task_struct
> > *task) return result;
> >  }
> >
> > +static void freezer_work_fn(struct work_struct *work)
> > +{
> > + struct freezer *freezer;
> > + unsigned long delay_jiffies = 0;
> > +
> > + freezer = container_of(work, struct freezer,

> > freezer_work.work);
> > + spin_lock_irq(&freezer->lock);
>
> So you switch interrupts off every once a while simply to
> freeze/unfreeze your cgroup.  That's going to affect the scheduler
> quite a bit, isn't it?
>
consider we are only doing this at long interval, perhaps the impact is
less. on the other side, the race is harmless as I mentioned in the
previous patch review, i could do without the lock.
> > + /* toggle between THAWED and FROZEN state. */
> > + update_if_frozen(freezer->css.cgroup, freezer);
>
> What about using a local variable cgroup = freezer->css.cgroup ?
>
sure, i can change that.
> > + switch (freezer->state) {
> > + case CGROUP_FROZEN:
> > + case CGROUP_FREEZING:
> > +  delay_jiffies = msecs_to_jiffies((100 -
> > freezer->duty.ratio) *
> > +
> > freezer->duty.period_pct_ms);
> > +  unfreeze_cgroup(freezer->css.cgroup, freezer);
> > +  break;
> > + case CGROUP_THAWED:
> > +  delay_jiffies =
> > msecs_to_jiffies(freezer->duty.ratio *
> > +
> > freezer->duty.period_pct_ms);
> > +  try_to_freeze_cgroup(freezer->css.cgroup, freezer);
> > +  break;
> > + default:
> > +  BUG();
>
> Don't crash the kernel if you don't have to.
>
ok. good point.
> > + }
> > +
> > + schedule_delayed_work(&freezer->freezer_work,
> > delay_jiffies);
>
> What if this work happens to run during system suspend/resume?
>
> > + spin_unlock_irq(&freezer->lock);
> > +}
> > +

> >  /*
> >   * cgroups_write_string() limits the size of freezer state strings
> > to
> >   * CGROUP_LOCAL_BUFFER_SIZE
> > @@ -144,13 +192,19 @@ static struct cgroup_subsys_state
> > *freezer_create(struct cgroup_subsys *ss,
> >   spin_lock_init(&freezer->lock);
> >   freezer->state = CGROUP_THAWED;
> > + freezer->duty.period_pct_ms = 10; /* default to 1 second
> > period */
>
> Why this number?
>
just a default value, keep it non-zero. also in the range that not too
much overhead in managing the toggling between FROZEN and THAWED.

> >   return &freezer->css;
> >  }
> >
> >  static void freezer_destroy(struct cgroup_subsys *ss,
> >         struct cgroup *cgroup)
> >  {
> > - kfree(cgroup_freezer(cgroup));
> > + struct freezer *freezer;
> > +
> > + freezer = cgroup_freezer(cgroup);
> > + if (freezer->duty.enabled)
> > +  cancel_delayed_work_sync(&freezer->freezer_work);
> > + kfree(freezer);
> >  }
> >
> >  /*
> > @@ -282,6 +336,16 @@ static int freezer_read(struct cgroup *cgroup,
> > struct cftype *cft, return 0;
> >  }
> >
> > +static u64 freezer_read_duty_ratio(struct cgroup *cgroup, struct
> > cftype *cft) +{
> > + return cgroup_freezer(cgroup)->duty.ratio;
> > +}
> > +
> > +static u64 freezer_read_period(struct cgroup *cgroup, struct
> > cftype *cft) +{
> > + return cgroup_freezer(cgroup)->duty.period_pct_ms / 10;
> > +}
> > +
> >  static int try_to_freeze_cgroup(struct cgroup *cgroup, struct
> > freezer *freezer) {

> >   struct cgroup_iter it;
> > @@ -347,20 +411,56 @@ out:
> >   return retval;
> >  }
> >
>
> It would be _nice_ to add a kerneldoc comment explaining the purpose
> of the new function, right?
>
> > +static void freezer_check_duty_cycling(struct freezer *freezer,
> > int set_state) +{
> > + if (freezer->duty.enabled) {
> > +  if (freezer->duty.ratio && freezer->duty.ratio <
> > 100)
> > +   return;
> > +  pr_debug("stopping duty ratio mode\n");
> > +  cancel_delayed_work_sync(&freezer->freezer_work);
> > +  freezer->duty.enabled = 0;
> > +  if (set_state) {
> > +   update_if_frozen(freezer->css.cgroup,
> > freezer);
> > +   if (freezer->duty.ratio == 100)
> > +
> > try_to_freeze_cgroup(freezer->css.cgroup,
> > +       freezer);
> > +   else if (freezer->duty.ratio == 0)
> > +
> > unfreeze_cgroup(freezer->css.cgroup, freezer);
> > +   else
> > +    BUG();
>
> Again, don't crash the kernel if you don't have to.
>
ditto.
> > +  }
> > + } else if (freezer->duty.ratio < 100 &&
> > freezer->duty.ratio) {
> > +  pr_debug("starting duty ratio mode\n");
> > +  INIT_DELAYED_WORK(&freezer->freezer_work,
> > freezer_work_fn);
> > +  freezer->duty.enabled = 1;
> > +  schedule_delayed_work(&freezer->freezer_work, 0);
> > + }
> > +}
> > +
> >  static int freezer_write(struct cgroup *cgroup,
> >       struct cftype *cft,
> >       const char *buffer)

> > {
> > int retval;
> > enum freezer_state goal_state;
> > + struct freezer *freezer;
> >
> > - if (strcmp(buffer, freezer_state_strs[CGROUP_THAWED]) == 0)
> > + freezer = cgroup_freezer(cgroup);
> > + if (strcmp(buffer, freezer_state_strs[CGROUP_THAWED]) ==
> > 0) { goal_state = CGROUP_THAWED;
> > - else if (strcmp(buffer, freezer_state_strs[CGROUP_FROZEN])
> > == 0)
> > + freezer->duty.ratio = 0;
> > + } else if (strcmp(buffer,
> > freezer_state_strs[CGROUP_FROZEN]) == 0) {
> > + freezer->duty.ratio = 100;
> > goal_state = CGROUP_FROZEN;
> > + }
> > else
> > return -EINVAL;
> >
> > + /* we should stop duty ratio toggling if user wants to
> > + * force change to a valid state.
> > + */
> > + freezer_check_duty_cycling(freezer, 0);
>
> Doesn't that need to be under the lock?
>
i guess so.
> > +
> > if (!cgroup_lock_live_group(cgroup))
> > return -ENODEV;
> > retval = freezer_change_state(cgroup, goal_state);
> > @@ -368,12 +468,61 @@ static int freezer_write(struct cgroup
> > *cgroup, return retval;
> > }
> >
>
> A kerneldoc comment would also be nice to have here.
>
ok, i will add that.
> > +static int freezer_write_param(struct cgroup *cgroup, struct
> > cftype *cft,
> > + u64 val)
> > +{
> > + struct freezer *freezer;
> > + int ret = 0;
> > +
> > + freezer = cgroup_freezer(cgroup);

```
> > + if (!cgroup_lock_live_group(cgroup))
> > +  return -ENODEV;
> > +
> > + spin_lock_irq(&freezer->lock);
> > + switch (cft->private) {
> > + case FREEZER_DUTY_RATIO:
> > +  if (val > 100) {
> > +   ret = -EINVAL;
> > +   goto exit;
> > +  }
> > +  freezer->duty.ratio = val;
> > +  break;
> > + case FREEZER_PERIOD:
> > +  if (!val) {
> > +   ret = -EINVAL;
> > +   goto exit;
> > +  }
> > +  freezer->duty.period_pct_ms = val * 10;
> > +  break;
> > + default:
> > +  BUG();
>
> Rule: Use BUG() only if the consequences of not doing so would be
> _worse_ than a kernel crash.
>
thanks for the tip, will fix.
> > + }
> > + spin_unlock_irq(&freezer->lock);
> > + freezer_check_duty_cycling(freezer, 1);
> > +exit:
> > + cgroup_unlock();
> > + return ret;
> > +}
> > +
> >  static struct cftype files[] = {
> >   {
> >    .name = "state",
> >    .read_seq_string = freezer_read,
> >    .write_string = freezer_write,
> >   },
> > + {
> > +  .name = "frozen_time_percentage",
> > +  .private = FREEZER_DUTY_RATIO,
> > +  .read_u64 = freezer_read_duty_ratio,
> > +  .write_u64 = freezer_write_param,
> > + },
> > + {
> > +  .name = "period_second",
```

> > +  .private = FREEZER_PERIOD,
> > +  .read_u64 = freezer_read_period,
> > +  .write_u64 = freezer_write_param,
> > + },
> > +
> >  };
> >
> >  static int freezer_populate(struct cgroup_subsys *ss, struct
> > cgroup *cgroup)
>
> Well, quite frankly, I'm not going to take this patch unless it gets
> an ACK from the scheduler people (which I'm guessing is not going to
> happen before hell freezes over).
>
> IOW, please find a better way to address the issue at hand.
>
We do have a real need that there is no exist feature in the kernel can provide solution for. You want ACK from scheduler people because it has impact on disabling irq? or you think scheduler should be the one that provide the solution. I did try cpu subsystem, but it seems to be limited to RT and certain scheduling policy e.g. RR and FIFO.

thanks,

Jacob

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