Subject: Re: [PATCH] introduce task cgroup v2
Posted by Li Zefan on Tue, 10 Jun 2008 05:22:07 GMT
View Forum Message <> Reply to Message

KOSAKI Motohiro wrote:
> ChangeLog
> ==
> v1 -> v2
> o implement can_attach() and attach() method.
> o remvoe can_fork member from cgroup_subsys.
> instead, copy_process() call task_cgroup_can_fork() directly.
> o fixed dead lock bug.
> o add Documentation/contoroller/task.txt
>

I think Documentation/kernel-parameters.txt should also be updated.

 cgroup_disable= [KNL] Disable a particular controller
  Format: {name of the controller(s) to disable}
  {Currently supported controllers - "memory"}

>
> benefit
> ==
> 1. prevent fork bomb.
>
> We already have "/prox/sys/kernel/threads-max".
> but it isn't perfect solution.
> because threads-max prevent any process creation.
> then, System-administrator can't login and restore trouble.
>
> task limit cgroup is better solution.
> it can prevent fork by network service daemon, but allow fork by interactive process.
>
>
> 2. help implement batch processing
>
> in general, batch processing environment need support #task limit.
> this patch help implement it.
>
>
> usage
> ==
>
> # mount -t cgroup -o task none /dev/cgroup
> # mkdir /dev/cgroup/foo
> # cd /dev/cgroup/foo
> # ls

> notify_on_release task.max_tasks task.nr_tasks tasks
> # echo 100 > task.max_tasks
> # echo $$ > tasks
> # fork_bomb 1000 & <- try create 1000 process
> # pgrep fork_bomb | wc -l
> 98
>
>
> Signed-off-by: KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com>
> CC: Li Zefan <lizf@cn.fujitsu.com>
> CC: Paul Menage <menage@google.com>
>
> ---
> Documentation/controllers/task.txt | 18 +++
> include/linux/cgroup_subsys.h | 4
> include/linux/cgroup_task.h | 33 +++++
> init/Kconfig | 10 +
> kernel/Makefile | 1
> kernel/cgroup_task.c | 213 ++++++++++++++++++++++++++++++++++++
> kernel/fork.c | 4
> 7 files changed, 283 insertions(+)
>
> Index: b/init/Kconfig
> ===
> --- a/init/Kconfig
> +++ b/init/Kconfig
> @@ -289,6 +289,16 @@ config CGROUP_DEBUG
>
>   Say N if unsure
>
> +config CGROUP_TASK
> + bool "Simple number of task accounting cgroup subsystem"
> + depends on CGROUPS && EXPERIMENTAL
> + default n
> + help
> +   Provides a simple number of task accounting cgroup subsystem for

should use TAB

> +   prevent fork bomb.
> +
> +   Say N if unsure
> +
> config CGROUP_NS
>   bool "Namespace cgroup subsystem"
>   depends on CGROUPS
> Index: b/kernel/Makefile
> ===

> --- a/kernel/Makefile
> +++ b/kernel/Makefile
> @@ -46,6 +46,7 @@ obj-$(CONFIG_KEXEC) += kexec.o
> obj-$(CONFIG_COMPAT) += compat.o
> obj-$(CONFIG_CGROUPS) += cgroup.o
> obj-$(CONFIG_CGROUP_DEBUG) += cgroup_debug.o
> +obj-$(CONFIG_CGROUP_TASK) += cgroup_task.o
> obj-$(CONFIG_CPUSETS) += cpuset.o
> obj-$(CONFIG_CGROUP_NS) += ns_cgroup.o
> obj-$(CONFIG_UTS_NS) += utsname.o
> Index: b/kernel/cgroup_task.c
> ===
> --- /dev/null
> +++ b/kernel/cgroup_task.c
> @@ -0,0 +1,213 @@
> +/* cgroup_task.c - #task control group
> + *
> + * Copyright (C) 2008 KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com>
> + *
> + * 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/err.h>
> +#include <linux/cgroup_task.h>
> +
> +struct task_cgroup {
> + struct cgroup_subsys_state css;
> + /*
> + * the counter to account for number of thread.
> + */
> + int max_tasks;
> + int nr_tasks;
> +
> + spinlock_t lock;
> +};
> +
> +struct cgroup_subsys task_cgroup_subsys __read_mostly;
> +
> +static inline struct task_cgroup *task_cgroup_from_task(struct task_struct *p)

```
> +{
> + return container_of(task_subsys_state(p, task_cgroup_subsys_id),
> +    struct task_cgroup, css);
> +}
> +
> +static struct task_cgroup *task_cgroup_from_cgrp(struct cgroup *cgrp)
> +{
> + return container_of(cgroup_subsys_state(cgrp,
> +    task_cgroup_subsys_id), struct task_cgroup,
> +    css);
> +}
> +
> +static int task_cgroup_max_tasks_write(struct cgroup *cgrp,
> +           struct cftype *cftype,
> +           s64 new_max_tasks)
> +{
> + struct task_cgroup *taskcg;
> + int ret = -EBUSY;
> +
> + if ((new_max_tasks > INT_MAX) ||
> +     (new_max_tasks < -1))
> +  return -EINVAL;
> +
> + taskcg = task_cgroup_from_cgrp(cgrp);
> +
> + spin_lock(&taskcg->lock);
> + if (taskcg->nr_tasks <= new_max_tasks) {
> +  taskcg->max_tasks = new_max_tasks;
> +  ret = 0;
> + }
> + spin_unlock(&taskcg->lock);
> +
> + return ret;
> +}
> +
> +static s64 task_cgroup_max_tasks_read(struct cgroup *cgrp, struct cftype *cft)
> +{
> + s64 max_tasks;
> + struct task_cgroup *taskcg = task_cgroup_from_cgrp(cgrp);
> +
> + spin_lock(&taskcg->lock);
> + max_tasks = taskcg->max_tasks;
> + spin_unlock(&taskcg->lock);
> +
> + return max_tasks;
> +}
> +
> +static s64 task_cgroup_nr_tasks_read(struct cgroup *cgrp, struct cftype *cft)
```

```
> +{
> + s64 nr_tasks;
> + struct task_cgroup *taskcg = task_cgroup_from_cgrp(cgrp);
> +
> + spin_lock(&taskcg->lock);
> + nr_tasks = taskcg->nr_tasks;
> + spin_unlock(&taskcg->lock);
> +
> + return nr_tasks;
> +}
> +
> +static struct cftype task_cgroup_files[] =  {
> + {
> +  .name = "max_tasks",
> +  .write_s64 = task_cgroup_max_tasks_write,
> +  .read_s64  = task_cgroup_max_tasks_read,
> + },
> + {
> +  .name = "nr_tasks",
> +  .read_s64  = task_cgroup_nr_tasks_read,
> + },
> +};
> +
> +static struct cgroup_subsys_state *task_cgroup_create(struct cgroup_subsys *ss,
> +         struct cgroup *cgrp)
> +{
> + struct task_cgroup *taskcg;
> +
> + taskcg = kzalloc(sizeof(struct task_cgroup), GFP_KERNEL);
> + if (!taskcg)
> +  return ERR_PTR(-ENOMEM);
> +
> + taskcg->max_tasks = -1;   /* infinite */
> + spin_lock_init(&taskcg->lock);
> +
> + return &taskcg->css;
> +}
> +
> +static void task_cgroup_destroy(struct cgroup_subsys *ss, struct cgroup *cgrp)
> +{
> + kfree(cgrp->subsys[task_cgroup_subsys_id]);
```

You should kfree task_cgroup* instead of cgroup_subsys_state*.

```
> +}
> +
> +
> +static int task_cgroup_can_attach(struct cgroup_subsys *ss,
```

```
> +      struct cgroup *cgrp, struct task_struct *tsk)
> +{
> + struct task_cgroup *taskcg = task_cgroup_from_cgrp(cgrp);
> + int ret = 0;
> +
> + spin_lock(&taskcg->lock);
> + if (taskcg->nr_tasks == taskcg->max_tasks)
> +  ret = -EBUSY;
> + spin_unlock(&taskcg->lock);
> +
> + return ret;
> +}
> +
> +static void task_cgroup_attach(struct cgroup_subsys *ss, struct cgroup *cgrp,
> +          struct cgroup *old_cgrp, struct task_struct *tsk)
> +{
> + struct task_cgroup *old_taskcg = task_cgroup_from_cgrp(old_cgrp);
> + struct task_cgroup *new_taskcg = task_cgroup_from_cgrp(cgrp);
> +
> + spin_lock(&old_taskcg->lock);
> + old_taskcg->nr_tasks--;
> + spin_unlock(&old_taskcg->lock);
> +
> + spin_lock(&new_taskcg->lock);
> + new_taskcg->nr_tasks++;
> + spin_unlock(&new_taskcg->lock);
> +}
> +
> +
> +static int task_cgroup_populate(struct cgroup_subsys *ss,
> +     struct cgroup *cgrp)
> +{
> + return cgroup_add_files(cgrp, ss, task_cgroup_files,
> +     ARRAY_SIZE(task_cgroup_files));
> +}
> +
> +int task_cgroup_can_fork(struct task_struct *task)
> +{
> + struct task_cgroup *taskcg;
> + int max_tasks;
> + int ret = 0;
> +
> + if (task_cgroup_subsys.disabled)
> +  return 0;
> +
> + taskcg = task_cgroup_from_task(task);
> +
> + spin_lock(&taskcg->lock);
```

```
> + max_tasks = taskcg->max_tasks;
> +
> + if ((max_tasks >= 0) &&
> +     (taskcg->nr_tasks >= max_tasks))
> +  ret = -EAGAIN;
> + else
> +  taskcg->nr_tasks++;
> + spin_unlock(&taskcg->lock);
> +
> + return ret;
> +}
> +
> +static void task_cgroup_exit(struct cgroup_subsys *ss, struct task_struct *task)
> +{
> + struct task_cgroup *taskcg;
> +
> + if (task_cgroup_subsys.disabled)
> +  return;
> +
> + taskcg = task_cgroup_from_task(task);
> +
> + spin_lock(&taskcg->lock);
> + taskcg->nr_tasks--;
> + spin_unlock(&taskcg->lock);
> +}
> +
> +
> +struct cgroup_subsys task_cgroup_subsys = {
> + .name = "task",
> + .subsys_id = task_cgroup_subsys_id,
> + .create = task_cgroup_create,
> + .destroy = task_cgroup_destroy,
> + .can_attach = task_cgroup_can_attach,
> + .attach = task_cgroup_attach,
> + .populate = task_cgroup_populate,
> + .exit = task_cgroup_exit,
> + .early_init = 0,
> +};
> +
> Index: b/kernel/fork.c
> ===================================================================
> --- a/kernel/fork.c
> +++ b/kernel/fork.c
> @@ -54,6 +54,7 @@
>  #include <linux/tty.h>
>  #include <linux/proc_fs.h>
>  #include <linux/blkdev.h>
> +#include <linux/cgroup_task.h>
```

>
> #include <asm/pgtable.h>
> #include <asm/pgalloc.h>
> @@ -920,6 +921,8 @@ static struct task_struct *copy_process(
>   p->user != current->nsproxy->user_ns->root_user)
>   goto bad_fork_free;
>   }
> + if (task_cgroup_can_fork(p))
> + goto bad_fork_free;

If task_cgroup_can_fork() returns 0, but copy_process() fails afterwords,
taskcg->nr_tasks will be in a wrong state.

>
>   atomic_inc(&p->user->__count);
>   atomic_inc(&p->user->processes);
> @@ -994,6 +997,7 @@ static struct task_struct *copy_process(
>   p->io_context = NULL;
>   p->audit_context = NULL;
>   cgroup_fork(p);
> +
> #ifdef CONFIG_NUMA
>   p->mempolicy = mpol_dup(p->mempolicy);
>   if (IS_ERR(p->mempolicy)) {
> Index: b/include/linux/cgroup_subsys.h
> ===
> --- a/include/linux/cgroup_subsys.h
> +++ b/include/linux/cgroup_subsys.h
> @@ -48,3 +48,7 @@ SUBSYS(devices)
> #endif
>
> /* */
> +
> +#ifdef CONFIG_CGROUP_TASK
> +SUBSYS(task_cgroup)
> +#endif
> Index: b/include/linux/cgroup_task.h
> ===
> --- /dev/null
> +++ b/include/linux/cgroup_task.h
> @@ -0,0 +1,33 @@
> +/* cgroup_task.h - #task control group
> + *
> + * Copyright (C) 2008 KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com>
> + *
> + * 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.
> + */
> +
> +#ifndef _LINUX_CGROUP_TASK_H
> +#define _LINUX_CGROUP_TASK_H
> +
> +#ifdef CONFIG_CGROUP_TASK
> +int task_cgroup_can_fork(struct task_struct *task);
> +
> +#else /*CONFIG_CGROUP_TASK*/
> +
> +static inline int task_cgroup_can_fork(struct task_struct *task)
> +{
> + return 0;
> +}
> +
> +#endif
> +
> +
> +#endif
> +
> Index: b/Documentation/controllers/task.txt
> ===
> --- /dev/null
> +++ b/Documentation/controllers/task.txt
> @@ -0,0 +1,18 @@
> +Task limit Controller
> +
> +1. Description
> +
> +Implement a cgroup to track number of tasks (process and thread).
> +this feature is useful for prevent fork bomb attack.
> +
> +
> +2. User Interface
> +
> +ristrict number of tasks to 100.
> +
> + # echo 100 > /cgroups/foo/task.max_tasks
> +
> +
> +display current number of tasks.
> +

> + # cat > /cgroups/foo/task.nr_tasks
>
>
>
>

_____

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