Subject: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by KOSAKI Motohiro on Thu, 05 Jun 2008 04:43:06 GMT
View Forum Message <> Reply to Message

Hi

I create new cgroup of number task restriction.
Please any comments!

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.

restrict of cgroup is better solution.
it can prevent fork by network service daemon, but allow fork interactive operation.

2. help implement batch processing

in general, batch environment need support #task restriction.
my 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
# fork_bomb 1000 &  <- try create 1000 process
# pgrep fork_bomb|wc -l
98
```

future work
==========
discussion cgroup guys more.

Signed-off-by: KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com>
CC: Li Zefan <lizf@cn.fujitsu.com>
CC: Paul Menage <menage@google.com>

---
 include/linux/cgroup.h | 5 -
 include/linux/cgroup_subsys.h | 4
 init/Kconfig | 10 ++
 kernel/Makefile | 1
 kernel/cgroup.c | 16 +++
 kernel/cgroup_task.c | 185 ++
 kernel/fork.c | 5 -
 7 files changed, 222 insertions(+), 4 deletions(-)

Index: b/include/linux/cgroup.h
===
--- a/include/linux/cgroup.h
+++ b/include/linux/cgroup.h
@@ -27,7 +27,7 @@ extern int cgroup_init(void);
 extern void cgroup_init_smp(void);
 extern void cgroup_lock(void);
 extern void cgroup_unlock(void);
-extern void cgroup_fork(struct task_struct *p);
+extern int cgroup_fork(struct task_struct *p);
 extern void cgroup_fork_callbacks(struct task_struct *p);
 extern void cgroup_post_fork(struct task_struct *p);
 extern void cgroup_exit(struct task_struct *p, int run_callbacks);
@@ -299,6 +299,7 @@ struct cgroup_subsys {
      struct cgroup *cgrp, struct task_struct *tsk);
  void (*attach)(struct cgroup_subsys *ss, struct cgroup *cgrp,
     struct cgroup *old_cgrp, struct task_struct *tsk);
+ int (*can_fork)(struct cgroup_subsys *ss, struct task_struct *task);
  void (*fork)(struct cgroup_subsys *ss, struct task_struct *task);
  void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
  int (*populate)(struct cgroup_subsys *ss,
@@ -381,7 +382,7 @@ int cgroup_attach_task(struct cgroup *,
 static inline int cgroup_init_early(void) { return 0; }
 static inline int cgroup_init(void) { return 0; }
 static inline void cgroup_init_smp(void) {}
-static inline void cgroup_fork(struct task_struct *p) {}
+static inline int cgroup_fork(struct task_struct *p) { return 0; }
 static inline void cgroup_fork_callbacks(struct task_struct *p) {}
 static inline void cgroup_post_fork(struct task_struct *p) {}
 static inline void cgroup_exit(struct task_struct *p, int callbacks) {}
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
+   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.c
===================================================================
--- a/kernel/cgroup.c
+++ b/kernel/cgroup.c
@@ -2719,13 +2719,27 @@ static struct file_operations proc_cgrou
  * At the point that cgroup_fork() is called, 'current' is the parent
  * task, and the passed argument 'child' points to the child task.
  */
-void cgroup_fork(struct task_struct *child)
+int cgroup_fork(struct task_struct *child)
 {
+ int i;
+ int ret;
+
+ for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
+  struct cgroup_subsys *ss = subsys[i];
+  if (ss->can_fork) {
```

```
+   ret = ss->can_fork(ss, child);
+   if (ret)
+    return ret;
+  }
+ }
+
  task_lock(current);
  child->cgroups = current->cgroups;
  get_css_set(child->cgroups);
  task_unlock(current);
  INIT_LIST_HEAD(&child->cg_list);
+
+ return 0;
 }

 /**
Index: b/kernel/cgroup_task.c
===================================================================
--- /dev/null
+++ b/kernel/cgroup_task.c
@@ -0,0 +1,185 @@
+/* cgroup_task.c - #task control group
+ *
+ * Copyright: 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/res_counter.h>
+#include <linux/cgroup.h>
+#include <linux/err.h>
+
+
+
+
+struct cgroup_subsys task_cgroup_subsys __read_mostly;
+
+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;
+};
+
+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 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 int task_cgroup_max_tasks_write(struct cgroup *cgrp,
+           struct cftype *cftype,
+           s64 max_tasks)
+{
+ struct task_cgroup *taskcg;
+
+ if ((max_tasks > INT_MAX) ||
+     (max_tasks < INT_MIN))
+  return -EINVAL;
+
+ taskcg = task_cgroup_from_cgrp(cgrp);
+
+ spin_lock(&taskcg->lock);
+ if (max_tasks < taskcg->nr_tasks)
+  return -EBUSY;
+ taskcg->max_tasks = max_tasks;
+ spin_unlock(&taskcg->lock);
+
+ return 0;
+}
+
+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]);
+}
+
+
+static int task_cgroup_populate(struct cgroup_subsys *ss,
+    struct cgroup *cgrp)
+{
+ if (task_cgroup_subsys.disabled)
+  return 0;
+
+ return cgroup_add_files(cgrp, ss, task_cgroup_files,
+    ARRAY_SIZE(task_cgroup_files));
+}
+
+static int  task_cgroup_can_fork(struct cgroup_subsys *ss,
+     struct task_struct *task)
+{
+ struct task_cgroup *taskcg;
+ int ret = 0;
+
+ if (task_cgroup_subsys.disabled)
+  return 0;
+
+ taskcg = task_cgroup_from_task(task);
+
+ spin_lock(&taskcg->lock);
+ if (taskcg->nr_tasks == taskcg->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,
+ .populate = task_cgroup_populate,
+ .can_fork = task_cgroup_can_fork,
+ .exit = task_cgroup_exit,
+ .early_init = 0,
+};
+
+
Index: b/kernel/fork.c
===================================================================
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -993,7 +993,9 @@ static struct task_struct *copy_process(
  p->cap_bset = current->cap_bset;
  p->io_context = NULL;
  p->audit_context = NULL;
- cgroup_fork(p);
+ if (cgroup_fork(p))
+  goto bad_fork_cleanup_delayacct;
+
 #ifdef CONFIG_NUMA
  p->mempolicy = mpol_dup(p->mempolicy);
   if (IS_ERR(p->mempolicy)) {
@@ -1257,6 +1259,7 @@ bad_fork_cleanup_policy:
 bad_fork_cleanup_cgroup:
 #endif
  cgroup_exit(p, cgroup_callbacks_done);
+bad_fork_cleanup_delayacct:
  delayacct_tsk_free(p);
  if (p->binfmt)
   module_put(p->binfmt->module);
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

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by KAMEZAWA Hiroyuki on Thu, 05 Jun 2008 05:09:46 GMT
View Forum Message <> Reply to Message

On Thu, 05 Jun 2008 13:43:06 +0900
KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com> wrote:
> +static int task_cgroup_max_tasks_write(struct cgroup *cgrp,
> + struct cftype *cftype,
> + s64 max_tasks)
> +{
> + struct task_cgroup *taskcg;
> +
> + if ((max_tasks > INT_MAX) ||
> + (max_tasks < INT_MIN))
> + return -EINVAL;

should be (max_tasks > INT_MAX) || (max_tasks < -1)) ?

> +
> + taskcg = task_cgroup_from_cgrp(cgrp);
> +
> + spin_lock(&taskcg->lock);
> + if (max_tasks < taskcg->nr_tasks)
> + return -EBUSY;
> + taskcg->max_tasks = max_tasks;
> + spin_unlock(&taskcg->lock);
This will cause dead lock.

And it seems this doesn't handle "attach failure".

It will be helpful Documentation somewhere.

Thanks,
-Kame

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by Daisuke Nishimura on Thu, 05 Jun 2008 05:23:20 GMT
View Forum Message <> Reply to Message

Hi.

I think this would be usefull.

On Thu, 05 Jun 2008 13:43:06 +0900, KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com> wrote:
> Index: b/kernel/cgroup_task.c
> ===
> --- /dev/null
> +++ b/kernel/cgroup_task.c
> @@ -0,0 +1,185 @@
> +/* cgroup_task.c - #task control group
> + *
> + * Copyright: 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/res_counter.h>
I don't think it's needed.
Or, are you planning to implement this feature by using res_counter?

> +static int task_cgroup_max_tasks_write(struct cgroup *cgrp,
> +   struct cftype *cftype,
> +   s64 max_tasks)
> +{
> + struct task_cgroup *taskcg;
> +
> + if ((max_tasks > INT_MAX) ||
> +   (max_tasks < INT_MIN))
> + return -EINVAL;
> +
> + taskcg = task_cgroup_from_cgrp(cgrp);

> +
> + spin_lock(&taskcg->lock);
> + if (max_tasks < taskcg->nr_tasks)
> + return -EBUSY;
need spin_unlock().

Thanks,
Daisuke Nishimura.

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

---

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by Li Zefan on Thu, 05 Jun 2008 05:47:39 GMT
View Forum Message <> Reply to Message

KOSAKI Motohiro wrote:
> Hi
>
> I create new cgroup of number task restriction.
> Please any comments!
>
>
>
> 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.
>
> restrict of cgroup is better solution.
> it can prevent fork by network service daemon, but allow fork interactive operation.
>
>
> 2. help implement batch processing
>
> in general, batch environment need support #task restriction.
> my 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
> # fork_bomb 1000 & <- try create 1000 process
> # pgrep fork_bomb|wc -l
> 98
>

You are not handling task migration between groups, i.e.:
 # echo $$ > /cgroup/1/tasks
 # echo $$ > /cgroup/2/tasks

>
> future work
> ==================================
> discussion cgroup guys more.
>
>
> Signed-off-by: KOSAKI Motohiro <kosaki.motohiro@jp.fujitsu.com>
> CC: Li Zefan <lizf@cn.fujitsu.com>
> CC: Paul Menage <menage@google.com>
>
> ---
> include/linux/cgroup.h | 5 -
> include/linux/cgroup_subsys.h | 4
> init/Kconfig | 10 ++
> kernel/Makefile | 1
> kernel/cgroup.c | 16 +++
> kernel/cgroup_task.c | 185 ++
> kernel/fork.c | 5 -
> 7 files changed, 222 insertions(+), 4 deletions(-)
>
> Index: b/include/linux/cgroup.h
> ===
> --- a/include/linux/cgroup.h
> +++ b/include/linux/cgroup.h
> @@ -27,7 +27,7 @@ extern int cgroup_init(void);
> extern void cgroup_init_smp(void);
> extern void cgroup_lock(void);
> extern void cgroup_unlock(void);
> -extern void cgroup_fork(struct task_struct *p);
> +extern int cgroup_fork(struct task_struct *p);
> extern void cgroup_fork_callbacks(struct task_struct *p);

```
>  extern void cgroup_post_fork(struct task_struct *p);
>  extern void cgroup_exit(struct task_struct *p, int run_callbacks);
> @@ -299,6 +299,7 @@ struct cgroup_subsys {
>       struct cgroup *cgrp, struct task_struct *tsk);
>   void (*attach)(struct cgroup_subsys *ss, struct cgroup *cgrp,
>     struct cgroup *old_cgrp, struct task_struct *tsk);
> + int  (*can_fork)(struct cgroup_subsys *ss, struct task_struct *task);
>   void (*fork)(struct cgroup_subsys *ss, struct task_struct *task);
>   void (*exit)(struct cgroup_subsys *ss, struct task_struct *task);
>   int (*populate)(struct cgroup_subsys *ss,
> @@ -381,7 +382,7 @@ int cgroup_attach_task(struct cgroup *,
>  static inline int cgroup_init_early(void) { return 0; }
>  static inline int cgroup_init(void) { return 0; }
>  static inline void cgroup_init_smp(void) {}
> -static inline void cgroup_fork(struct task_struct *p) {}
> +static inline int  cgroup_fork(struct task_struct *p) { return 0; }
>  static inline void cgroup_fork_callbacks(struct task_struct *p) {}
>  static inline void cgroup_post_fork(struct task_struct *p) {}
>  static inline void cgroup_exit(struct task_struct *p, int callbacks) {}
> 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
> +   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.c
> ===================================================================
> --- a/kernel/cgroup.c
> +++ b/kernel/cgroup.c
> @@ -2719,13 +2719,27 @@ static struct file_operations proc_cgrou
>   * At the point that cgroup_fork() is called, 'current' is the parent
>   * task, and the passed argument 'child' points to the child task.
>   */
> -void cgroup_fork(struct task_struct *child)
> +int cgroup_fork(struct task_struct *child)
>  {
> + int i;
> + int ret;
> +
> + for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
```

Maybe you need to add this in cgroup_init_subsys():

```
-       need_forkexit_callback |= ss->fork || ss->exit;
+       need_forkexit_callback |= ss->fork || ss->exit || ss->can_fork;
```

and then we can do:

```
if (need_forkexit_callback) {
 for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
  ...
 }
}
```

```
> +  struct cgroup_subsys *ss = subsys[i];
> +  if (ss->can_fork) {
> +   ret = ss->can_fork(ss, child);
> +   if (ret)
> +    return ret;
> +  }
> + }
> +
>   task_lock(current);
>   child->cgroups = current->cgroups;
>   get_css_set(child->cgroups);
>   task_unlock(current);
>   INIT_LIST_HEAD(&child->cg_list);
> +
> + return 0;
```

> }

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by Dhaval Giani on Thu, 05 Jun 2008 09:27:45 GMT
View Forum Message <> Reply to Message

> +static int task_cgroup_max_tasks_write(struct cgroup *cgrp,
> + struct cftype *cftype,
> + s64 max_tasks)
> +{
> + struct task_cgroup *taskcg;
> +
> + if ((max_tasks > INT_MAX) ||
> + (max_tasks < INT_MIN))

It should be < -1 I think.

> + return -EINVAL;
> +
> + taskcg = task_cgroup_from_cgrp(cgrp);
> +
> + spin_lock(&taskcg->lock);
> + if (max_tasks < taskcg->nr_tasks)
> + return -EBUSY;

Shouldn't you drop the lock here?

> + taskcg->max_tasks = max_tasks;
> + spin_unlock(&taskcg->lock);
> +
> + return 0;
> +}

How does this controller affect performance? Do you have some numbers?

Thanks,
--
regards,
Dhaval

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by KOSAKI Motohiro on Thu, 05 Jun 2008 10:51:35 GMT
View Forum Message <> Reply to Message

Hi

Thank you for careful review.

>> + struct task_cgroup *taskcg;
>> +
>> + if ((max_tasks > INT_MAX) ||
>> + (max_tasks < INT_MIN))
>
> It should be < -1 I think.

OK.
I'll fix it at next post.

>> + spin_lock(&taskcg->lock);
>> + if (max_tasks < taskcg->nr_tasks)
>> + return -EBUSY;
>
> Shouldn't you drop the lock here?

you are right.
Thanks.

> How does this controller affect performance? Do you have some numbers?

No.
but I have plan to mesure it at nearly future.
Thanks.

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

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)

Posted by KOSAKI Motohiro on Thu, 05 Jun 2008 10:53:50 GMT
View Forum Message <> Reply to Message

> You are not handling task migration between groups, i.e.:
> # echo $$ > /cgroup/1/tasks
> # echo $$ > /cgroup/2/tasks

Oh, nice point out.
Thanks!

> Maybe you need to add this in cgroup_init_subsys():
>
> - need_forkexit_callback |= ss->fork || ss->exit;
> + need_forkexit_callback |= ss->fork || ss->exit || ss->can_fork;
>
> and then we can do:
>
> if (need_forkexit_callback) {
> for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {

good idea :)
I'll fix it at next post.

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

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by KOSAKI Motohiro on Thu, 05 Jun 2008 10:56:32 GMT
View Forum Message <> Reply to Message

Hi Nishimura-san,

Thanks good point out.

>> +#include <linux/res_counter.h>
> I don't think it's needed.
> Or, are you planning to implement this feature by using res_counter?

your are right.
my early version use res_counter, but it isn't used currently.

>> + spin_lock(&taskcg->lock);
>> + if (max_tasks < taskcg->nr_tasks)

>> + return -EBUSY;
> need spin_unlock().

Yes, of cource.
I'll fix it.

Thanks!

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

---

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by Daniel Hokka Zakrisso on Thu, 05 Jun 2008 21:52:36 GMT
View Forum Message <> Reply to Message

KOSAKI Motohiro wrote:
> Hi
>
> I create new cgroup of number task restriction.
> Please any comments!

Would it make more sense to implement this as part of an rlimit subsystem, which also supports limiting e.g. address space, CPU time, number of open files, etc.? If we create one subsystem per resource, I'm afraid we're going to see quite some time spent in all those loops, and the options for cgroupfs is going to become pretty long if you want to exclude just one or two of the subsystems for one particular mount point.

--
Daniel Hokka Zakrisson

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

---

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by Paul Menage on Thu, 05 Jun 2008 22:04:12 GMT
View Forum Message <> Reply to Message

Hi Kosaki,

The basic idea of a task-limiting subsystem is good, thanks.

On Wed, Jun 4, 2008 at 9:43 PM, KOSAKI Motohiro
<kosaki.motohiro@jp.fujitsu.com> wrote:
> --- a/kernel/cgroup.c
> +++ b/kernel/cgroup.c
> @@ -2719,13 +2719,27 @@ static struct file_operations proc_cgrou
> * At the point that cgroup_fork() is called, 'current' is the parent
> * task, and the passed argument 'child' points to the child task.
> */
> -void cgroup_fork(struct task_struct *child)
> +int cgroup_fork(struct task_struct *child)
> {
> +   int i;
> +   int ret;
> +
> +   for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
> +   struct cgroup_subsys *ss = subsys[i];
> +   if (ss->can_fork) {
> +   ret = ss->can_fork(ss, child);
> +   if (ret)
> +   return ret;
> +   }
> +   }
> +
>   task_lock(current);
>   child->cgroups = current->cgroups;
>   get_css_set(child->cgroups);
>   task_unlock(current);
>   INIT_LIST_HEAD(&child->cg_list);
> +
> +   return 0;
> }

I don't think this is the right way to handle this check. This isn't a
generic control groups callback, it's one that specific for a
particular subsystem. So the right way to handle it is to call
task_cgroup_can_fork() from the same place that the RLIM_NPROC limit
is checked.

If it later turned out that multiple cgroup subsystems wanted to be
able to prevent forking, then it might make sense to have a generic
cgroup callback, but for just one subsystem it's cleaner to call
directly.

> +
> +static int task_cgroup_populate(struct cgroup_subsys *ss,
> +   struct cgroup *cgrp)
> +{

```
> +       if (task_cgroup_subsys.disabled)
> +               return 0;
```

I don't think you should need this check - if the subsystem is
disabled, it'll never be mounted in the first place.

Paul

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by KOSAKI Motohiro on Sat, 07 Jun 2008 06:46:56 GMT
View Forum Message <> Reply to Message

Hi

> Hi Kosaki,
>
> The basic idea of a task-limiting subsystem is good, thanks.

Thanks.

```
> > -void cgroup_fork(struct task_struct *child)
> > +int cgroup_fork(struct task_struct *child)
> >  {
> > +       int i;
> > +       int ret;
> > +
> > +       for (i = 0; i < CGROUP_SUBSYS_COUNT; i++) {
> > +               struct cgroup_subsys *ss = subsys[i];
> > +               if (ss->can_fork) {
> > +                       ret = ss->can_fork(ss, child);
> > +                       if (ret)
> > +                               return ret;
> > +               }
> > +       }
> > +
> >         task_lock(current);
> >         child->cgroups = current->cgroups;
> >         get_css_set(child->cgroups);
> >         task_unlock(current);
> >         INIT_LIST_HEAD(&child->cg_list);
> > +
```

> > +	return 0;
> > }
>
> I don't think this is the right way to handle this check. This isn't a
> generic control groups callback, it's one that specific for a
> particular subsystem. So the right way to handle it is to call
> task_cgroup_can_fork() from the same place that the RLIM_NPROC limit
> is checked.
>
> If it later turned out that multiple cgroup subsystems wanted to be
> able to prevent forking, then it might make sense to have a generic
> cgroup callback, but for just one subsystem it's cleaner to call
> directly.

OK.

> > +static int task_cgroup_populate(struct cgroup_subsys *ss,
> > +				struct cgroup *cgrp)
> > +{
> > +	if (task_cgroup_subsys.disabled)
> > +		return 0;
>
> I don't think you should need this check - if the subsystem is
> disabled, it'll never be mounted in the first place.

to be honest, I did copy&past it from memcontrol.c ;)
Thanks good opinion.

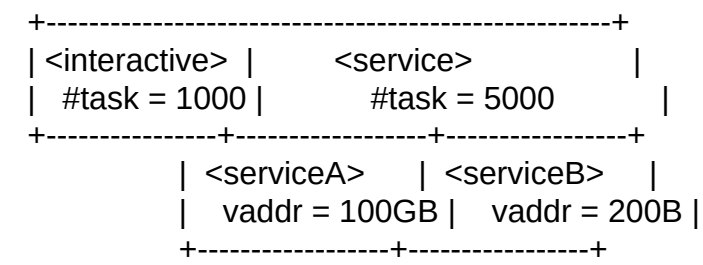
Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by KOSAKI Motohiro on Sat, 07 Jun 2008 07:41:02 GMT
View Forum Message <> Reply to Message

> > I create new cgroup of number task restriction.
> > Please any comments!
>
> Would it make more sense to implement this as part of an rlimit subsystem,
> which also supports limiting e.g. address space, CPU time, number of open
> files, etc.? If we create one subsystem per resource, I'm afraid we're
> going to see quite some time spent in all those loops, and the options for

> cgroupfs is going to become pretty long if you want to exclude just one or
> two of the subsystems for one particular mount point.

Good quistion.
task cgroup often different scope against memory, cputime, etc..

example.
(<> mean cgroup name)



tus, we have 2 choice.

1. separate 2 cgroup.
2. implement hierarchy.

I afraid to 2 cause perfomance degression.
tus, I choiced 1.

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

---

Subject: Re: [RFC][PATCH] introduce task cgroup (#task restrictioon for prevent fork bomb by cgroup)
Posted by Paul Menage on Sat, 07 Jun 2008 09:12:49 GMT
View Forum Message <> Reply to Message

On Thu, Jun 5, 2008 at 2:52 PM, Daniel Hokka Zakrisson <daniel@hozac.com> wrote:
> Would it make more sense to implement this as part of an rlimit subsystem,
> which also supports limiting e.g. address space, CPU time, number of open
> files, etc.? If we create one subsystem per resource, I'm afraid we're
> going to see quite some time spent in all those loops, and the options for
> cgroupfs is going to become pretty long if you want to exclude just one or
> two of the subsystems for one particular mount point.
>

In general, most of the cgroup loops that depend on the number of

available subsystems are control path (fork/exit/task move) rather
than data path.

If these loops ever did become an issue it wouldn't be hard to cache
which subsystems had callbacks of a given type, and thus make the loop
overheap be linear in the number of subsystems actually using a given
callback, rather than the number of registered subsystems. I've not
done this yet since it didn't seem worth the complexity, but if the
number of subsystems grows from what we have now it might be
worthwhile.

Having a single rlimit subsystem seems a bit inflexible, since it
would require anyone that wanted to track one resource to have to pay
the overhead of tracking all resources. By splitting into separate
subsystems, it would be possible to just pay the overhead for the
resource tracking that you actually care about.

You mention that you think it makes the set of mount arguments more
complex, but if you have middleware that can limit resource X, adding
X to the list of subsystems at the point when a cgroupfs is mounted
seems simple compared to the rest of the structure required to actual
set limits and monitor usage for X.

Paul