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

_____

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