Subject: [RFC][PATCH 14/14] Enable cloning pid namespace
Posted by Sukadev Bhattiprolu on Wed, 21 Mar 2007 03:24:38 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [RFC][PATCH 14/14] Enable cloning pid namespace

When clone() is invoked with CLONE_NEWPID, create a new pid namespace
and then create a new struct pid for the new process. Allocate pid_ts
for the new process in the new pid namespace and all ancestor pid
namespaces. Make the newly cloned process the session and process group
leader.

Since the primary pid namespace is special and expected to be the first
entry in pid_nrs list, preserve the order of pid namespaces when
cloning without CLONE_NEWPID.

TODO (partial list:)

- Exclude some CLONE_* flags with CLONE_NEWPID (eg does it make
  sense when both CLONE_THREAD and CLONE_NEWPID are set) ?
- Add a privilege check for CLONE_NEWPID
- procfs support - mainly remounting /proc in child namespaces (work
  is in progress)
- Maybe use list_head to represent pid_nrs list so preserving
  order of namespaces across clone() is more efficient.
- Add CONFIG_PID_NS (experimental) before sending to LKML

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 include/linux/pid.h | 2
 kernel/fork.c | 16 ++++---
 kernel/pid.c | 115 +++---
 3 files changed, 122 insertions(+), 11 deletions(-)

Index: lx26-21-rc3-mm2/kernel/pid.c
==========
--- lx26-21-rc3-mm2.orig/kernel/pid.c 2007-03-20 15:55:52.000000000 -0700
+++ lx26-21-rc3-mm2/kernel/pid.c 2007-03-20 15:55:52.000000000 -0700
@@ -307,11 +307,90 @@ struct pid_namespace *pid_ns(struct pid
  return pid_nr->pid_ns;
 }

-struct pid *alloc_pid(void)
+void append_pid_nr(struct pid *pid, struct pid_nr *pid_nr)
+{
+ struct pid_nr *last = NULL;
+ struct hlist_node *pos;

```
+ struct pid_nr *xpid_nr;
+
+ hlist_for_each_entry(xpid_nr, pos, &pid->pid_nrs, node)
+  last = xpid_nr;
+
+ if (last)
+  hlist_add_after(&last->node, &pid_nr->node);
+ else
+  hlist_add_head(&pid_nr->node, &pid->pid_nrs);
+}
+
+static int dup_parent_pid_nrs(struct pid *new_pid)
+{
+ struct pid_nr *pid_nr;
+ struct pid_nr *ppid_nr;
+ struct hlist_node *pos;
+ struct pid *parent_pid = task_pid(current);
+
+ hlist_for_each_entry(ppid_nr, pos, &parent_pid->pid_nrs, node) {
+  pid_nr = alloc_pid_nr(ppid_nr->pid_ns, new_pid);
+  if (!pid_nr)
+   return -ENOMEM;        // caller should call free_pid()
+  /*
+   * The first pid_nr on new_pid->pid_nrs list must be from the
+   * same pid namespace as the first element on parent_pid->
+   * pid_nrs list (since the first element defines the clone-pid
+   * namespace of a struct pid).
+   *
+   * So try to preserve the order of namespaces in parent and
+   * child for now. To do this with a hlist (pid->pid_nrs is
+   * a hlist_head), we need to find and append entries to the
+   * tail of the list.
+   *
+   * We can improve performance by making pid->pid_nrs a
+   * list_head rather than a hlist_head and then using
+   * list_add_tail(). Costs an extra word in struct pid :-(.
+   */
+  append_pid_nr(new_pid, pid_nr);
+ }
+
+ return 0;
+}
+
+static struct pid_namespace *clone_pid_ns(struct task_struct *reaper)
+{
+ struct pid_namespace *ns;
+ int i;
+
```

```
+ ns = kmalloc(sizeof(struct pid_namespace), GFP_KERNEL);
+ if (!ns)
+  return ns;
+
+ kref_init(&ns->kref);
+
+ atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
+ ns->pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
+ if (!ns->pidmap[0].page) {
+  kfree(ns);
+  return NULL;
+ }
+
+ set_bit(0, ns->pidmap[0].page);
+
+ for (i = 1; i < PIDMAP_ENTRIES; i++) {
+  atomic_set(&ns->pidmap[i].nr_free, BITS_PER_PAGE);
+  ns->pidmap[i].page = NULL;
+ }
+ ns->last_pid = 0;
+ ns->child_reaper = reaper;
+
+ return ns;
+}
+
+struct pid *alloc_pid(unsigned long flags, struct task_struct *reaper)
 {
  struct pid *pid;
  enum pid_type type;
  struct pid_nr *pid_nr;
+ struct pid_namespace *new_pid_ns = NULL;
+ int rc;

  pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
  if (!pid)
@@ -324,16 +403,30 @@ struct pid *alloc_pid(void)

  INIT_HLIST_HEAD(&pid->pid_nrs);

- pid_nr = alloc_pid_nr(task_pid_ns(current), pid);
- if (!pid_nr)
+ rc = dup_parent_pid_nrs(pid);
+ if (rc)
   goto out_free_pid;

- hlist_add_head(&pid_nr->node, &pid->pid_nrs);
+ if (flags & CLONE_NEWPID) {
+  new_pid_ns = clone_pid_ns(reaper);
```

```
+  if (!new_pid_ns)
+   goto out_free_pid;
+
+  pid_nr = alloc_pid_nr(new_pid_ns, pid);
+  if (!pid_nr)
+   goto out_free_pid_ns;
+
+  hlist_add_head(&pid_nr->node, &pid->pid_nrs);
+ }

  return pid;

+out_free_pid_ns:
+ if (new_pid_ns)
+  put_pid_ns(new_pid_ns);
+
 out_free_pid:
- kmem_cache_free(pid_cachep, pid);
+ free_pid(pid);
  return NULL;
 }

@@ -470,9 +563,21 @@ EXPORT_SYMBOL_GPL(find_get_pid);

 void free_pid_ns(struct kref *kref)
 {
+ int i;
+ int nr_free;
  struct pid_namespace *ns;

  ns = container_of(kref, struct pid_namespace, kref);
+
+ BUG_ON(ns == &init_pid_ns);
+
+ for (i = 0; i < PIDMAP_ENTRIES; i++) {
+  nr_free = atomic_read(&ns->pidmap[i].nr_free);
+  BUG_ON(nr_free != BITS_PER_PAGE);
+
+  if (ns->pidmap[i].page)
+   kfree(ns->pidmap[i].page);
+ }
  kfree(ns);
 }

Index: lx26-21-rc3-mm2/include/linux/pid.h
===================================================================
--- lx26-21-rc3-mm2.orig/include/linux/pid.h 2007-03-20 15:55:52.000000000 -0700
+++ lx26-21-rc3-mm2/include/linux/pid.h 2007-03-20 15:55:52.000000000 -0700
```

```
@@ -115,7 +115,7 @@ extern struct pid *find_ge_pid(int nr);
 extern void free_pid_nr(struct pid_nr *pid_nr);
 extern struct pid_nr *alloc_pid_nr(struct pid_namespace *pid_ns,
    struct pid *pid);
-extern struct pid *alloc_pid(void);
+extern struct pid *alloc_pid(unsigned long clone_flags, struct task_struct *p);
 extern void FASTCALL(free_pid(struct pid *pid));
 extern pid_t pid_nr(struct pid *pid);

Index: lx26-21-rc3-mm2/kernel/fork.c
===================================================================
--- lx26-21-rc3-mm2.orig/kernel/fork.c 2007-03-20 15:55:52.000000000 -0700
+++ lx26-21-rc3-mm2/kernel/fork.c 2007-03-20 15:55:52.000000000 -0700
@@ -1037,7 +1037,7 @@ static struct task_struct *copy_process(
  if (unlikely(idle_process == COPY_IDLE_PROCESS))
   pid = &init_struct_pid;
  else {
-  pid = alloc_pid();
+  pid = alloc_pid(clone_flags, p);
   if (!pid)
    goto bad_fork_put_binfmt_module;
  }
@@ -1263,11 +1263,17 @@ static struct task_struct *copy_process(
  tracehook_init_task(p);

  if (thread_group_leader(p)) {
+  struct pid *pgrp = task_pgrp(current);
+  struct pid *session = task_session(current);
+
+  if (unlikely(clone_flags & CLONE_NEWPID))
+   pgrp = session = pid;
+
   p->signal->tty = current->signal->tty;
-  p->signal->pgrp = process_group(current);
-  set_signal_session(p->signal, process_session(current));
-  attach_pid(p, PIDTYPE_PGID, task_pgrp(current));
-  attach_pid(p, PIDTYPE_SID, task_session(current));
+  p->signal->pgrp = pid_nr(pgrp);
+  set_signal_session(p->signal, pid_nr(session));
+  attach_pid(p, PIDTYPE_PGID, pgrp);
+  attach_pid(p, PIDTYPE_SID, session);

   list_add_tail_rcu(&p->tasks, &init_task.tasks);
   __get_cpu_var(process_counts)++;
```

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