Subject: [RFC][PATCH 11/16] Enable cloning pid namespace
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:13:22 GMT
View Forum Message <> Reply to Message

Subject: Enable cloning pid namespace

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

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_t's 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 active pid namespace is special and expected to be the first entry in pid->upid_list, preserve the order of pid namespaces when cloning without CLONE_NEWPID.

TODO (partial list:)

- Identify clone flags that should not be specified with CLONE_NEWPID
  and return -EINVAL from copy_process(), if they are specified. (eg:
  CLONE_THREAD|CLONE_NEWPID ?)
- Add a privilege check for CLONE_NEWPID

Changelog:

2.6.21-mm2-pidns3:
- 'struct upid' used to be called 'struct pid_nr' and a list of these
  were hanging off of 'struct pid'. So, we renamed 'struct pid_nr'
  and now hold them in a statically sized array in 'struct pid' since
  the number of 'struct upid's for a process is known at process-
  creation time

2.6.21-mm2:
- [Serge Hallyn] Terminate other processes in pid ns when reaper is
  exiting.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

```
---
 include/linux/pid.h           |    3
 include/linux/pid_namespace.h |    5 -
 init/Kconfig                  |    9 +
 kernel/exit.c                 |   14 ++
 kernel/fork.c                 |   17 ++-
 kernel/pid.c                  |  209 +++++++++++++++++++++++++++++++++++++++++----
 6 files changed, 227 insertions(+), 30 deletions(-)
```

```
Index: lx26-21-mm2/kernel/pid.c
===================================================================
--- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:50.000000000 -0700
+++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:52.000000000 -0700
@@ -32,6 +32,7 @@
 #define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
 static struct hlist_head *pid_hash;
 static int pidhash_shift;
+static struct kmem_cache *pid1_cachep;
 static struct kmem_cache *pid_cachep;
 struct upid init_struct_upid = INIT_STRUCT_UPID;
 struct pid init_struct_pid = INIT_STRUCT_PID;
@@ -250,7 +251,12 @@ static struct upid *pid_active_upid(stru
 /*
  * Return the active pid namespace of the process @pid.
  *
- * Note: At present, there is only one pid namespace (init_pid_ns).
+ * Note:
+ * To avoid having to use an extra pointer in struct pid to keep track
+ * of active pid namespace, dup_struct_pid() maintains the order of
+ * entries in 'pid->upid_list' such that the youngest (or the 'active')
+ * pid namespace is the first entry and oldest (init_pid_ns) is the last
+ * entry in the list.
  */
 struct pid_namespace *pid_active_pid_ns(struct pid *pid)
 {
@@ -259,6 +265,64 @@ struct pid_namespace *pid_active_pid_ns(
 EXPORT_SYMBOL_GPL(pid_active_pid_ns);

 /*
+ * Return the parent pid_namespace of the active pid namespace of @tsk.
+ *
+ * Note:
+ *  Refer to function header of pid_active_pid_ns() for information on
+ *  the order of entries in pid->upid_list. Based on the order, the parent
+ *  pid namespace of the active pid namespace of @tsk is just the second
+ *  entry in the process's pid->upid_list.
+ *
+ *  Parent pid namespace of init_pid_ns is init_pid_ns itself.
+ */
+static struct pid_namespace *task_active_pid_ns_parent(struct task_struct *tsk)
+{
+ int idx = 0;
+ struct pid *pid = task_pid(tsk);
+
+ if (pid->num_upids > 1)
+  idx++;
```

```
+
+ return pid->upid_list[idx].pid_ns;
+}
+
+/*
+ * Return the child reaper of @tsk.
+ *
+ * Normally the child reaper of @tsk is simply the child reaper
+ * the active pid namespace of @tsk.
+ *
+ * But if @tsk is itself child reaper of a namespace, NS1, its child
+ * reaper depends on the caller.  If someone from an ancestor namespace
+ * or, if the reaper himself is asking, return the reaper of our parent
+ * namespace.
+ *
+ * If someone from namespace NS1 (other than reaper himself) is asking,
+ * return reaper of NS1.
+ */
+struct task_struct *task_child_reaper(struct task_struct *tsk)
+{
+ struct pid_namespace *tsk_ns = task_active_pid_ns(tsk);
+ struct task_struct *tsk_reaper = tsk_ns->child_reaper;
+ struct pid_namespace *my_ns;
+
+ /*
+  * TODO: Check if we need a lock here. ns->child_reaper
+  *   can change in do_exit() when reaper is exiting.
+  */
+
+ if (tsk != tsk_reaper)
+  return tsk_reaper;
+
+ my_ns = task_active_pid_ns(current);
+ if (my_ns != tsk_ns || current == tsk)
+  return task_active_pid_ns_parent(tsk)->child_reaper;
+
+ return tsk_reaper;
+}
+EXPORT_SYMBOL(task_child_reaper);
+
+/*
  * Return the pid_t by which the process @pid is known in the pid
  * namespace @ns.
  *
@@ -301,15 +365,78 @@ pid_t pid_to_nr(struct pid *pid)
 }
 EXPORT_SYMBOL_GPL(pid_to_nr);
```

```
+#ifdef CONFIG_PID_NS
+static int init_ns_pidmap(struct pid_namespace *ns)
+{
+ int i;
+
+ 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)
+  return -ENOMEM;
+
+ 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;
+ }
+ return 0;
+}
+
+static struct pid_namespace *alloc_pid_ns(void)
+{
+ struct pid_namespace *ns;
+ int rc;
+
+ ns = kzalloc(sizeof(struct pid_namespace), GFP_KERNEL);
+ if (!ns)
+  return NULL;
+
+ rc = init_ns_pidmap(ns);
+ if(rc) {
+  kfree(ns);
+  return NULL;
+ }
+
+ kref_init(&ns->kref);
+
+ return ns;
+}
+
+#else
+
+static int alloc_pid_ns()
+{
+ static int warned;
+
+ if (!warned) {
+  printk(KERN_INFO "WARNING: CLONE_NEWPID disabled\n");
```

```
+  warned = 1;
+ }
+ return 0;
+}
+#endif  /*CONFIG_PID_NS*/
+
+void toss_pid(struct pid *pid)
+{
+ if (pid->num_upids == 1)
+  kmem_cache_free(pid1_cachep, pid);
+ else {
+  kfree(pid->upid_list);
+  kmem_cache_free(pid_cachep, pid);
+ }
+}
+
 fastcall void put_pid(struct pid *pid)
 {
  if (!pid)
   return;

  if ((atomic_read(&pid->count) == 1) ||
-      atomic_dec_and_test(&pid->count)) {
-  kmem_cache_free(pid_cachep, pid);
- }
+      atomic_dec_and_test(&pid->count))
+  toss_pid(pid);
 }
 EXPORT_SYMBOL_GPL(put_pid);

@@ -345,15 +472,28 @@ static struct pid *alloc_struct_pid(int
  enum pid_type type;
  struct upid *upid_list;
  void *pid_end;
+ struct kmem_cache *cachep = pid1_cachep;

- /* for now we only support one pid namespace */
- BUG_ON(num_upids != 1);
- pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
+ if (num_upids > 1)
+  cachep = pid_cachep;
+
+ pid = kmem_cache_alloc(cachep, GFP_KERNEL);
  if (!pid)
   return NULL;

- pid_end = (void *)pid + sizeof(struct pid);
- pid->upid_list = (struct upid *)pid_end;
```

```
+ if (num_upids == 1) {
+  pid_end = (void *)pid + sizeof(struct pid);
+  pid->upid_list = (struct upid *)pid_end;
+ } else {
+  int upid_list_size = num_upids * sizeof(struct upid);
+
+  upid_list = kzalloc(upid_list_size, GFP_KERNEL);
+  if (!upid_list) {
+   kmem_cache_free(pid_cachep, pid);
+   return NULL;
+  }
+  pid->upid_list = upid_list;
+ }

  atomic_set(&pid->count, 1);
  pid->num_upids = num_upids;
@@ -364,7 +504,8 @@ static struct pid *alloc_struct_pid(int
  return pid;
 }

-struct pid *dup_struct_pid(enum copy_process_type copy_src)
+struct pid *dup_struct_pid(enum copy_process_type copy_src,
+  unsigned long clone_flags, struct task_struct *new_task)
 {
  int rc;
  int i;
@@ -379,20 +520,38 @@ struct pid *dup_struct_pid(enum copy_pro
   return &init_struct_pid;

  num_upids = parent_pid->num_upids;
+ if (clone_flags & CLONE_NEWPID)
+  num_upids++;

  pid = alloc_struct_pid(num_upids);
  if (!pid)
   return NULL;

  upid = &pid->upid_list[0];
+
+ if (clone_flags & CLONE_NEWPID) {
+  struct pid_namespace *new_pid_ns = alloc_pid_ns();
+
+  if (!new_pid_ns)
+   goto out_free_pid;
+
+  new_pid_ns->child_reaper = new_task;
+  rc = init_upid(upid, pid, new_pid_ns);
+  if (rc < 0)
```

```
+   goto out_free_pid;
+  upid++;
+ }
+
  parent_upid = &parent_pid->upid_list[0];

- for (i = 0; i < num_upids; i++, upid++, parent_upid++) {
+ for (i = 0; i < parent_pid->num_upids; i++, upid++, parent_upid++) {
   rc = init_upid(upid, pid, parent_upid->pid_ns);
   if (rc < 0)
    goto out_free_pid;
  }

+ new_task->pid = pid_active_upid(pid)->nr;
+
  return pid;

 out_free_pid:
@@ -533,9 +692,21 @@ 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);
 }

@@ -566,14 +737,22 @@ void __init pidhash_init(void)

 void __init pidmap_init(void)
 {
- int pid_elem_size;
+ int pid1_elem_size;

  init_pid_ns.pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
```

```
  /* Reserve PID 0. We never call free_pidmap(0) */
  set_bit(0, init_pid_ns.pidmap[0].page);
  atomic_dec(&init_pid_ns.pidmap[0].nr_free);

- pid_elem_size = sizeof(struct pid) + sizeof(struct upid);
- pid_cachep = kmem_cache_create("pid+1upid", pid1_elem_size, 0,
- SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+ /*
+  * Cache for struct pids with more than one pid namespace
+  */
+ pid_cachep = KMEM_CACHE(pid, SLAB_PANIC);
+
+ /*
+  * Cache for struct pids with exactly one pid namespace
+  */
+ pid1_elem_size = sizeof(struct pid) + sizeof(struct upid);
+ pid1_cachep = kmem_cache_create("pid+1upid", pid1_elem_size, 0,
+    SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
 }
Index: lx26-21-mm2/include/linux/pid.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid.h 2007-05-22 16:59:49.000000000 -0700
+++ lx26-21-mm2/include/linux/pid.h 2007-05-22 16:59:52.000000000 -0700
@@ -118,7 +118,8 @@ extern struct pid *FASTCALL(find_pid(int
 extern struct pid *find_get_pid(int nr);
 extern struct pid *find_ge_pid(int nr);

-extern struct pid *dup_struct_pid(enum copy_process_type);
+extern struct pid *dup_struct_pid(enum copy_process_type,
+  unsigned long clone_flags, struct task_struct *new_task);
 extern void FASTCALL(free_pid(struct pid *pid));

 extern pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid);
Index: lx26-21-mm2/include/linux/pid_namespace.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:50.000000000 -0700
+++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 16:59:52.000000000 -0700
@@ -54,9 +54,6 @@ static inline struct pid_namespace *task
  return pid_active_pid_ns(task_pid(tsk));
 }

-static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
-{
- return task_active_pid_ns(tsk)->child_reaper;
-}
+extern struct task_struct *task_child_reaper(struct task_struct *tsk);

 #endif /* _LINUX_PID_NS_H */
```

```
Index: lx26-21-mm2/init/Kconfig
===================================================================
--- lx26-21-mm2.orig/init/Kconfig 2007-05-22 16:58:36.000000000 -0700
+++ lx26-21-mm2/init/Kconfig 2007-05-22 16:59:52.000000000 -0700
@@ -250,6 +250,15 @@ config UTS_NS
    vservers, to use uts namespaces to provide different
    uts info for different servers.  If unsure, say N.

+config PID_NS
+ depends on EXPERIMENTAL
+ bool "PID Namespaces"
+ default n
+ help
+   Support multiple PID namespaces. This allows containers, i.e.
+   vservers to use separate different PID namespaces to different
+   servers. If unsuare, say N.
+
 config AUDIT
  bool "Auditing support"
  depends on NET
Index: lx26-21-mm2/kernel/exit.c
===================================================================
--- lx26-21-mm2.orig/kernel/exit.c 2007-05-22 16:59:46.000000000 -0700
+++ lx26-21-mm2/kernel/exit.c 2007-05-22 16:59:52.000000000 -0700
@@ -866,6 +866,7 @@ fastcall NORET_TYPE void do_exit(long co
 {
  struct task_struct *tsk = current;
  int group_dead;
+ struct pid_namespace *pid_ns = task_active_pid_ns(tsk);

  profile_task_exit(tsk);

@@ -875,10 +876,15 @@ fastcall NORET_TYPE void do_exit(long co
   panic("Aiee, killing interrupt handler!");
  if (unlikely(!tsk->pid))
   panic("Attempted to kill the idle task!");
- if (unlikely(tsk == task_child_reaper(tsk))) {
-  if (task_active_pid_ns(tsk) != &init_pid_ns)
-   task_active_pid_ns(tsk)->child_reaper =
-    init_pid_ns.child_reaper;
+
+ /*
+  * Note that we cannot use task_child_reaper() here because
+  * it returns reaper for parent pid namespace if tsk is itself
+  * the reaper of the active pid namespace.
+  */
+ if (unlikely(tsk == pid_ns->child_reaper)) {
+  if (pid_ns != &init_pid_ns)
```

```
+   pid_ns->child_reaper = init_pid_ns.child_reaper;
   else
    panic("Attempted to kill init!");
  }
Index: lx26-21-mm2/kernel/fork.c
===================================================================
--- lx26-21-mm2.orig/kernel/fork.c 2007-05-22 16:59:49.000000000 -0700
+++ lx26-21-mm2/kernel/fork.c 2007-05-22 16:59:52.000000000 -0700
@@ -1026,14 +1026,13 @@ static struct task_struct *copy_process(
  if (p->binfmt && !try_module_get(p->binfmt->module))
   goto bad_fork_cleanup_put_domain;

- pid = dup_struct_pid(copy_src);
+ pid = dup_struct_pid(copy_src, clone_flags, p);
  if (!pid)
   goto bad_fork_put_binfmt_module;

  p->did_exec = 0;
  delayacct_tsk_init(p); /* Must remain after dup_task_struct() */
  copy_flags(clone_flags, p);
- p->pid = pid_to_nr(pid);

  INIT_LIST_HEAD(&p->children);
  INIT_LIST_HEAD(&p->sibling);
@@ -1255,11 +1254,17 @@ static struct task_struct *copy_process(
    __ptrace_link(p, current->parent);

   if (thread_group_leader(p)) {
+   struct pid *pgrp = task_pgrp(current);
+   struct pid *session = task_session(current);
+
+   if (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_to_nr(pgrp);
+   set_signal_session(p->signal, pid_to_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