Subject: [PATCH 11/17] Pid-NS(V3) Enable cloning pid namespace
Posted by Sukadev Bhattiprolu on Sat, 16 Jun 2007 23:02:37 GMT
View Forum Message <> Reply to Message

Subject: [PATCH 11/17] Pid-NS(V3) 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.

Pid namespaces can be nested and the nesting depth is statically limited
to MAX_NESTED_PID_NS (arbitrarily set to 4 at present).

The size of 'struct pid' is dependent on the the number of pid namespaces
the process exists in, so we use multiple pid-caches'. Only one pid cache
is created during system startup and this used by processes that exist
only in init_pid_ns.

When a process clones its pid namespace, we create additional pid caches
as necessary and use the pid cache to allocate 'struct pids' for that depth.

If the nesting level of pid namespace exceeds MAX_NESTED_PID_NS, we fail
the clone() system call with an -EINVAL error.

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:

2.6.22-rc4-mm2-pidns2:

- Place a compile-time limit on levels of pid namespace-nesting
  and use this limit to optimize 'struct pid' allocation.

2.6.22-rc4-mm2-pidns1:

- Optimize the 'struct pid' and 'pid->upid_list' memory allocation

for processes that can exist in pid namespaces upto 3 levels
deep (i.e init_pid_ns plus two).

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

- [Badari Pulavarty] Free new_pid_ns if init_upid() fails

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           |   33 +++++++-
 include/linux/pid_namespace.h |   23 +++++
 init/Kconfig                  |    9 ++
 kernel/exit.c                 |   14 ++-
 kernel/fork.c                 |   21 +++--
 kernel/pid.c                  |  170 +++++++++++++++++++++++++++++++++++++-----
 6 files changed, 236 insertions(+), 34 deletions(-)
```

Index: lx26-22-rc4-mm2/kernel/pid.c
==========
```
--- lx26-22-rc4-mm2.orig/kernel/pid.c 2007-06-16 02:08:32.000000000 -0700
+++ lx26-22-rc4-mm2/kernel/pid.c 2007-06-16 04:13:57.000000000 -0700
@@ -33,7 +33,6 @@

 static struct hlist_head *pid_hash;
 static int pidhash_shift;
-static struct kmem_cache *pid_cachep;
 struct pid init_struct_pid = INIT_STRUCT_PID;

 int pid_max = PID_MAX_DEFAULT;
@@ -46,6 +45,43 @@ int pid_max_max = PID_MAX_LIMIT;
 #define BITS_PER_PAGE  (PAGE_SIZE*8)
 #define BITS_PER_PAGE_MASK (BITS_PER_PAGE-1)

+#define MAX_NESTED_PID_NS 4
+
+static struct kmem_cache *pid_cacheps[MAX_NESTED_PID_NS+1];
+
+/* kmem_cache_create() requires cache names not be on stack */
+static char *pid_cache_names[MAX_NESTED_PID_NS+1] = {
```

```
+ NULL, "pid+1upid", "pid+2upid", "pid+3upid", "pid+4upid"
+};
+
+static int ensure_pid_cache_exists(int num_upids)
+{
+ struct kmem_cache *cachep;
+ int pid_elem_size;
+ char *cache_name;
+
+ if (num_upids == 0 || num_upids > MAX_NESTED_PID_NS)
+  return -EINVAL;
+
+ /* Does one already exist for this level of nesting ? */
+ if (pid_cacheps[num_upids])
+  return 0;
+
+ cache_name = pid_cache_names[num_upids];
+
+ pid_elem_size = sizeof(struct pid);
+ pid_elem_size += (num_upids - 1) * sizeof(struct upid);
+
+ cachep = kmem_cache_create(cache_name, pid_elem_size, 0,
+    SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+ if (!cachep)
+  return -ENOMEM;
+
+ pid_cacheps[num_upids] = cachep;
+
+ return 0;
+}
+
 static inline int mk_pid(struct pid_namespace *pid_ns,
   struct pidmap *map, int off)
 {
@@ -236,6 +272,61 @@ 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 ERR_PTR(-ENOMEM);
+
+ rc = init_ns_pidmap(ns);
+ if (rc) {
+  kfree(ns);
+  return NULL;
+ }
+
+ kref_init(&ns->kref);
+
+ return ns;
+}
+
+#else
+
+static struct pid_namespace *alloc_pid_ns(void)
+{
+ WARN_ON_ONCE(1);
+ return ERR_PTR(-EINVAL);
+}
+#endif  /*CONFIG_PID_NS*/
+
+static inline struct kmem_cache *select_pid_cache(int num_upids)
+{
+ BUG_ON(num_upids == 0 || num_upids > MAX_NESTED_PID_NS);
+ return pid_cacheps[num_upids];
+}
+
 fastcall void put_pid(struct pid *pid)
 {
  if (!pid)
```

```
@@ -243,7 +334,7 @@ fastcall void put_pid(struct pid *pid)

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

@@ -277,10 +368,9 @@ static struct pid *alloc_struct_pid(int
 {
  struct pid *pid;
  enum pid_type type;
+ struct kmem_cache *cachep = select_pid_cache(num_upids);

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

@@ -293,13 +383,15 @@ 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;
- int num_upids;
+ int cidx;  /* loop index for child upids */
+ int pidx;  /* loop index for parent upids */
+ int cnum_upids;  /* # of upids in child pid */
  struct pid *pid;
- struct upid *upid;
+ struct upid *child_upid;
  struct upid *parent_upid;
  struct pid *parent_pid = task_pid(current);

@@ -307,25 +399,50 @@ struct pid *dup_struct_pid(enum copy_pro
  if (unlikely(copy_src == COPY_IDLE_PROCESS))
   return &init_struct_pid;

- num_upids = parent_pid->num_upids;
+ cnum_upids = parent_pid->num_upids;
```

```
+ if (clone_flags & CLONE_NEWPID) {
+  cnum_upids++;
+  if ((rc = ensure_pid_cache_exists(cnum_upids)))
+   return ERR_PTR(rc);
+ }

- pid = alloc_struct_pid(num_upids);
+ pid = alloc_struct_pid(cnum_upids);
  if (!pid)
-  return NULL;
+  return ERR_PTR(-ENOMEM);

- for (i = 0; i < num_upids; i++) {
-  upid = &pid->upid_list[i];
-  parent_upid = &parent_pid->upid_list[i];
-  rc = init_upid(upid, pid, parent_upid->pid_ns);
+ cidx = pidx = 0;
+ child_upid = &pid->upid_list[cidx];
+
+ if (clone_flags & CLONE_NEWPID) {
+  struct pid_namespace *new_pid_ns = alloc_pid_ns();
+
+  if (IS_ERR(new_pid_ns))
+   goto out_free_pid;
+
+  new_pid_ns->child_reaper = new_task;
+  rc = init_upid(child_upid, pid, new_pid_ns);
+  if (rc < 0) {
+   put_pid_ns(new_pid_ns);
+   goto out_free_pid;
+  }
+  cidx++;
+ }
+
+ for (; cidx < cnum_upids; pidx++, cidx++) {
+  parent_upid = &parent_pid->upid_list[pidx];
+  child_upid = &pid->upid_list[cidx];
+
+  rc = init_upid(child_upid, pid, parent_upid->pid_ns);
   if (rc)
    goto out_free_pid;
  }
+ new_task->pid = pid_active_upid(pid)->nr;

  return pid;

 out_free_pid:
  free_pid(pid);
```

```
- return NULL;
+ return ERR_PTR(-ENOMEM);
 }

 struct pid * fastcall find_pid(int nr)
@@ -463,9 +580,21 @@ 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);
 }

@@ -501,5 +630,10 @@ void __init pidmap_init(void)
  set_bit(0, init_pid_ns.pidmap[0].page);
  atomic_dec(&init_pid_ns.pidmap[0].nr_free);

- pid_cachep = KMEM_CACHE(pid, SLAB_PANIC);
+ /*
+  * Create a pid cache for 'struct pids' with just one pid namespace.
+  * For nesting depth greater than 1, we create the pid cache(s) when
+  * we really need them (see dup_struct_pid()).
+  */
+ (void)ensure_pid_cache_exists(1);
 }
Index: lx26-22-rc4-mm2/include/linux/pid.h
===================================================================
--- lx26-22-rc4-mm2.orig/include/linux/pid.h 2007-06-16 02:08:32.000000000 -0700
+++ lx26-22-rc4-mm2/include/linux/pid.h 2007-06-16 02:37:16.000000000 -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(struct pid *pid);
@@ -136,7 +137,11 @@ extern pid_t pid_to_nr(struct pid *pid);
 /*
  * Return the active upid of the process @pid.
  *
- * Note: At present, there is only one upid corresponding to 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.
  */
 static inline struct upid *pid_active_upid(struct pid *pid)
 {
@@ -145,8 +150,6 @@ static inline struct upid *pid_active_up

 /*
  * Return the active pid namespace of the process @pid.
- *
- * Note: At present, there is only one pid namespace (init_pid_ns).
  */
 static inline struct pid_namespace *pid_active_pid_ns(struct pid *pid)
 {
@@ -154,6 +157,28 @@ static inline struct pid_namespace *pid_
 }

 /*
+ * 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 inline struct pid_namespace *pid_active_pid_ns_parent(struct pid *pid)
+{
+ extern struct pid_namespace init_pid_ns;
+ struct pid_namespace *ns = &init_pid_ns;
+
+ if (pid->upid_list[0].pid_ns != &init_pid_ns)
+  ns = pid->upid_list[1].pid_ns;
```

```
+
+ return ns;
+}
+
+/*
  * Return the pid_t by which the process @pid is known in the pid
  * namespace @ns.
  *
Index: lx26-22-rc4-mm2/include/linux/pid_namespace.h
===================================================================
--- lx26-22-rc4-mm2.orig/include/linux/pid_namespace.h 2007-06-16 02:08:32.000000000 -0700
+++ lx26-22-rc4-mm2/include/linux/pid_namespace.h 2007-06-16 02:37:16.000000000 -0700
@@ -55,9 +55,30 @@ static inline struct pid_namespace *task
  return pid_active_pid_ns(task_pid(tsk));
 }

+/*
+ * 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.
+ */
 static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
 {
- return task_active_pid_ns(tsk)->child_reaper;
+ struct task_struct *reaper;
+ struct pid *pid = task_pid(tsk);
+
+ reaper = pid_active_pid_ns(pid)->child_reaper;
+ if (tsk == reaper)
+  reaper = pid_active_pid_ns_parent(pid)->child_reaper;
+
+ return reaper;
 }

 #endif /* _LINUX_PID_NS_H */
Index: lx26-22-rc4-mm2/init/Kconfig
===================================================================
--- lx26-22-rc4-mm2.orig/init/Kconfig 2007-06-16 02:08:32.000000000 -0700
+++ lx26-22-rc4-mm2/init/Kconfig 2007-06-16 02:15:21.000000000 -0700
```

```
@@ -248,6 +248,15 @@ 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: lx26-22-rc4-mm2/kernel/exit.c
===================================================================
--- lx26-22-rc4-mm2.orig/kernel/exit.c 2007-06-16 02:08:32.000000000 -0700
+++ lx26-22-rc4-mm2/kernel/exit.c 2007-06-16 02:37:16.000000000 -0700
@@ -871,6 +871,7 @@ 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);

@@ -880,10 +881,15 @@ 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: lx26-22-rc4-mm2/kernel/fork.c
===================================================================
--- lx26-22-rc4-mm2.orig/kernel/fork.c 2007-06-16 02:08:32.000000000 -0700
+++ lx26-22-rc4-mm2/kernel/fork.c 2007-06-16 02:37:16.000000000 -0700
@@ -1027,14 +1027,15 @@ 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);
- if (!pid)
+ pid = dup_struct_pid(copy_src, clone_flags, p);
+ if (IS_ERR(pid)) {
+  retval = PTR_ERR(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);
  p->vfork_done = NULL;
@@ -1263,11 +1264,17 @@ 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
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers