Subject: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Pavel Emelianov on Tue, 15 Jan 2008 12:50:58 GMT
View Forum Message <> Reply to Message

We have one bit in the clone_flags left, so we won't be
able to create more namespaces after we make it busy.
Besides, for checkpoint/restart jobs we might want to
create tasks with pre-defined pids (virtual of course).
What else might be required from clone() - nobody knows.

This is an attempt to create a extendable API for clone.

I use the last bit in the clone_flags for CLONE_NEWCLONE.
When set it will denote that the child_tidptr is not a
pointer on the tid storage, but the pointer on the struct
long_clone_struct which currently looks like this:

```
struct long_clone_arg {
 int size;
};
```

When we want to add a new argument for clone we just put
it *at the end of this structure* and adjust the size.
The binary compatibility with older long_clone_arg-s is
facilitated with the clone_arg_has() macro.

Sure, we lose the ability to clone tasks with extended
argument and the CLONE_CHILD_SETTID/CLEARTID, but do we
really need this?

The same thing is about to be done for unshare - we can
add the second argument for it and iff the CLONE_NEWCLONE
is specified - try to use it. Binary compatibility with
the old ushare will be kept.

The new argument is pulled up to the create_new_namespaces
so that later we can easily use it w/o sending additional
patches.

This is a final, but a pre-review patch for sys_clone()
that I plan to send to Andrew before we go on developing
new namespaces.

Made against 2.6.24-rc5-mm1.

Signed-off-by: Pavel Emelyanov <xemul@openzv.org>

```
diff --git a/include/linux/nsproxy.h b/include/linux/nsproxy.h
index 0e66b57..e01de56 100644
--- a/include/linux/nsproxy.h
+++ b/include/linux/nsproxy.h
@@ -62,7 +62,8 @@ static inline struct nsproxy *task_nsproxy(struct task_struct *tsk)
  return rcu_dereference(tsk->nsproxy);
 }

-int copy_namespaces(unsigned long flags, struct task_struct *tsk);
+int copy_namespaces(unsigned long flags, struct task_struct *tsk,
+  struct long_clone_arg *carg);
 void exit_task_namespaces(struct task_struct *tsk);
 void switch_task_namespaces(struct task_struct *tsk, struct nsproxy *new);
 void free_nsproxy(struct nsproxy *ns);
diff --git a/include/linux/sched.h b/include/linux/sched.h
index e4d2a82..585a2b4 100644
--- a/include/linux/sched.h
+++ b/include/linux/sched.h
@@ -27,6 +27,23 @@
 #define CLONE_NEWUSER  0x10000000 /* New user namespace */
 #define CLONE_NEWPID  0x20000000 /* New pid namespace */
 #define CLONE_NEWNET  0x40000000 /* New network namespace */
+#define CLONE_NEWCLONE  0x80000000 /* Has an long_clone_arg */
+
+/*
+ * If the CLONE_NEWCLONE argument is specified, then the
+ * child_tidptr is expected to point to the struct long_clone_arg.
+ *
+ * This structure has a variable size, which is to be recorded
+ * in the size member. All other members a to be put after this.
+ * Never ... No. Always add new members only at its tail.
+ *
+ * The clone_arg_has() macro is responsible for checking whether
+ * the given arg has the desired member.
+ */
+
+struct long_clone_arg {
+ int size;
+};

 /*
  * Scheduling policies
@@ -40,6 +57,11 @@

 #ifdef __KERNEL__

+#define clone_arg_has(arg, member) ({ \
```

```
+  struct long_clone_arg *__carg = arg; \
+  (__carg->size >= offsetof(struct long_clone_arg, member) + \
+      sizeof(__carg->member)); })
+
 struct sched_param {
  int sched_priority;
 };
diff --git a/kernel/fork.c b/kernel/fork.c
index 8b558b7..f5895fc 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -967,6 +967,29 @@ static void rt_mutex_init_task(struct task_struct *p)
 #endif
 }

+static struct long_clone_arg *get_long_clone_arg(int __user *child_tidptr)
+{
+ int size;
+ struct long_clone_arg *carg;
+
+ if (get_user(size, child_tidptr))
+  return ERR_PTR(-EFAULT);
+
+ if (size > sizeof(struct long_clone_arg))
+  return ERR_PTR(-EINVAL);
+
+ carg = kzalloc(sizeof(struct long_clone_arg), GFP_KERNEL);
+ if (carg == NULL)
+  return ERR_PTR(-ENOMEM);
+
+ if (copy_from_user(carg, child_tidptr, size)) {
+  kfree(carg);
+  return ERR_PTR(-EFAULT);
+ }
+
+ return carg;
+}
+
 /*
  * This creates a new process as a copy of the old one,
  * but does not actually start it yet.
@@ -985,6 +1008,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
  int retval;
  struct task_struct *p;
  int cgroup_callbacks_done = 0;
+ struct long_clone_arg *carg = NULL;

  if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
```

```
   return ERR_PTR(-EINVAL);
@@ -1004,6 +1028,11 @@ static struct task_struct *copy_process(unsigned long clone_flags,
  if ((clone_flags & CLONE_SIGHAND) && !(clone_flags & CLONE_VM))
   return ERR_PTR(-EINVAL);

+ if ((clone_flags & CLONE_NEWCLONE) &&
+   (clone_flags & (CLONE_CHILD_SETTID |
+     CLONE_CHILD_CLEARTID)))
+  return ERR_PTR(-EINVAL);
+
  retval = security_task_create(clone_flags);
  if (retval)
   goto fork_out;
@@ -1132,8 +1161,15 @@ static struct task_struct *copy_process(unsigned long clone_flags,
  /* Perform scheduler related setup. Assign this task to a CPU. */
  sched_fork(p, clone_flags);

+ if (clone_flags & CLONE_NEWCLONE) {
+  carg = get_long_clone_arg(child_tidptr);
+  retval = PTR_ERR(carg);
+  if (IS_ERR(carg))
+   goto bad_fork_cleanup_policy;
+ }
+
  if ((retval = security_task_alloc(p)))
-  goto bad_fork_cleanup_policy;
+  goto bad_fork_cleanup_carg;
  if ((retval = audit_alloc(p)))
   goto bad_fork_cleanup_security;
  /* copy all the process information */
@@ -1151,7 +1187,7 @@ static struct task_struct *copy_process(unsigned long clone_flags,
   goto bad_fork_cleanup_signal;
  if ((retval = copy_keys(clone_flags, p)))
   goto bad_fork_cleanup_mm;
- if ((retval = copy_namespaces(clone_flags, p)))
+ if ((retval = copy_namespaces(clone_flags, p, carg)))
   goto bad_fork_cleanup_keys;
  retval = copy_thread(0, clone_flags, stack_start, stack_size, p, regs);
  if (retval)
@@ -1345,6 +1381,8 @@ bad_fork_cleanup_audit:
  audit_free(p);
 bad_fork_cleanup_security:
  security_task_free(p);
+bad_fork_cleanup_carg:
+ kfree(carg);
 bad_fork_cleanup_policy:
 #ifdef CONFIG_NUMA
  mpol_free(p->mempolicy);
```

```
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index f5d332c..b955196 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -48,7 +48,8 @@ static inline struct nsproxy *clone_nsproxy(struct nsproxy *orig)
  * leave it to the caller to do proper locking and attach it to task.
  */
 static struct nsproxy *create_new_namespaces(unsigned long flags,
-   struct task_struct *tsk, struct fs_struct *new_fs)
+   struct task_struct *tsk, struct fs_struct *new_fs,
+   struct long_clone_arg *carg)
 {
  struct nsproxy *new_nsp;
  int err;
@@ -119,7 +120,8 @@ out_ns:
  * called from clone.  This now handles copy for nsproxy and all
  * namespaces therein.
  */
-int copy_namespaces(unsigned long flags, struct task_struct *tsk)
+int copy_namespaces(unsigned long flags, struct task_struct *tsk,
+  struct long_clone_arg *carg)
 {
  struct nsproxy *old_ns = tsk->nsproxy;
  struct nsproxy *new_ns;
@@ -131,7 +133,8 @@ int copy_namespaces(unsigned long flags, struct task_struct *tsk)
  get_nsproxy(old_ns);

  if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
-    CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET)))
+    CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET |
+    CLONE_NEWCLONE)))
   return 0;

  if (!capable(CAP_SYS_ADMIN)) {
@@ -139,7 +142,7 @@ int copy_namespaces(unsigned long flags, struct task_struct *tsk)
   goto out;
  }

- new_ns = create_new_namespaces(flags, tsk, tsk->fs);
+ new_ns = create_new_namespaces(flags, tsk, tsk->fs, carg);
  if (IS_ERR(new_ns)) {
   err = PTR_ERR(new_ns);
   goto out;
@@ -191,7 +194,7 @@ int unshare_nsproxy_namespaces(unsigned long unshare_flags,
   return -EPERM;

  *new_nsp = create_new_namespaces(unshare_flags, current,
-    new_fs ? new_fs : current->fs);
```

```
+    new_fs ? new_fs : current->fs, NULL);
  if (IS_ERR(*new_nsp)) {
   err = PTR_ERR(*new_nsp);
   goto out;
```

_____

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