Subject: [PATCH 1/2] Extend sys_clone and sys_unshare system calls API
Posted by Pavel Emelianov on Wed, 16 Jan 2008 12:58:55 GMT
View Forum Message <> Reply to Message

There's only 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 given
pids (virtual of course). And nobody knows for sure what else might
be required from clone() in the future.

This is an attempt to create a extendable API for clone and unshare.
Actually this patch is a request for comment about the overall
design. If it will turn out to "look good", then we'll select some
better names for new flag and data types.

I use the last bit in the clone_flags for CLONE_LONGARG. When set it
will denote that the child_tidptr is not a pointer to a tid storage,
but the pointer to 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 helper macro
clone_arg_has(). It checks, that the desired member is provided from
the userspace.

Cedric implemented the same thing for unshare - he added the second
argument for it and iff the CLONE_NEWCLONE is specified - uses it.
Binary compatibility with the old ushare will be kept.

The new argument is to be pulled up to the create_new_namespaces,
which is done in a second patch.

This patch is made against the 2.6.24-rc6-mm1. It is tested to pass
more flags into create_new_namespaces.

Signed-off-by: Pavel Emelyanov <xemul@openzv.org>
Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
Reviewed-by: Serge Hallyn <serue@us.ibm.com>

---

diff --git a/include/linux/sched.h b/include/linux/sched.h
index 0a15018..f9332bb 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_LONGARG  0x80000000 /* Has an long_clone_arg */
+
+/*
+ * If the CLONE_LONGARG 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/include/linux/syscalls.h b/include/linux/syscalls.h
index 4c2577b..1cf3541 100644
--- a/include/linux/syscalls.h
+++ b/include/linux/syscalls.h
@@ -585,7 +585,7 @@ asmlinkage long compat_sys_newfstatat(unsigned int dfd, char __user *
filename,
           int flag);
 asmlinkage long compat_sys_openat(unsigned int dfd, const char __user *filename,
        int flags, int mode);
-asmlinkage long sys_unshare(unsigned long unshare_flags);
```

```
+asmlinkage long sys_unshare(unsigned long unshare_flags, int __user *flagptr);

 asmlinkage long sys_splice(int fd_in, loff_t __user *off_in,
       int fd_out, loff_t __user *off_out,
diff --git a/kernel/fork.c b/kernel/fork.c
index 19873c7..5e85567 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -967,6 +967,33 @@ static void rt_mutex_init_task(struct task_struct *p)
 #endif
 }

+static struct long_clone_arg *get_long_clone_arg(unsigned long flags,
+  int __user *child_tidptr)
+{
+ int size;
+ struct long_clone_arg *carg;
+
+ if (!(flags & CLONE_LONGARG))
+  return NULL;
+
+ 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 +1012,7 @@ 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;
```

```
  if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
   return ERR_PTR(-EINVAL);
@@ -1004,6 +1032,11 @@ 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_LONGARG) &&
+    (clone_flags & (CLONE_CHILD_SETTID |
+      CLONE_CHILD_CLEARTID)))
+  return ERR_PTR(-EINVAL);
+
  retval = security_task_create(clone_flags);
  if (retval)
   goto fork_out;
@@ -1141,8 +1174,14 @@ 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 ((retval = security_task_alloc(p)))
+ carg = get_long_clone_arg(clone_flags, child_tidptr);
+ if (IS_ERR(carg)) {
+  retval = PTR_ERR(carg);
   goto bad_fork_cleanup_policy;
+ }
+
+ if ((retval = security_task_alloc(p)))
+  goto bad_fork_cleanup_carg;
  if ((retval = audit_alloc(p)))
   goto bad_fork_cleanup_security;
  /* copy all the process information */
@@ -1355,6 +1394,8 @@ 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);
@@ -1672,7 +1713,7 @@ static int unshare_semundo(unsigned long unshare_flags, struct
sem_undo_list **n
  * constructed. Here we are modifying the current, active,
  * task_struct.
  */
-asmlinkage long sys_unshare(unsigned long unshare_flags)
+asmlinkage long sys_unshare(unsigned long unshare_flags, int __user *flagptr)
 {
  int err = 0;
```

```
  struct fs_struct *fs, *new_fs = NULL;
@@ -1681,6 +1722,7 @@ asmlinkage long sys_unshare(unsigned long unshare_flags)
  struct files_struct *fd, *new_fd = NULL;
  struct sem_undo_list *new_ulist = NULL;
  struct nsproxy *new_nsproxy = NULL;
+ struct long_clone_arg *carg;

  check_unshare_flags(&unshare_flags);

@@ -1689,11 +1731,17 @@ asmlinkage long sys_unshare(unsigned long unshare_flags)
  if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
     CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWUSER|
-    CLONE_NEWNET))
+    CLONE_NEWNET|CLONE_LONGARG))
   goto bad_unshare_out;

- if ((err = unshare_thread(unshare_flags)))
+ carg = get_long_clone_arg(unshare_flags, flagptr);
+ if (IS_ERR(carg)) {
+  err = PTR_ERR(carg);
   goto bad_unshare_out;
+ }
+
+ if ((err = unshare_thread(unshare_flags)))
+  goto bad_unshare_cleanup_carg;
  if ((err = unshare_fs(unshare_flags, &new_fs)))
   goto bad_unshare_cleanup_thread;
  if ((err = unshare_sighand(unshare_flags, &new_sigh)))
@@ -1763,6 +1811,8 @@ bad_unshare_cleanup_fs:
   put_fs_struct(new_fs);

 bad_unshare_cleanup_thread:
+bad_unshare_cleanup_carg:
+ kfree(carg);
 bad_unshare_out:
  return err;
 }
```

_____

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