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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Cedric Le Goater on Tue, 15 Jan 2008 13:38:06 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> 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.

hmm, I wonder how lkml@ will react to this. do we have
similar apis in the kernel ?

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

not in the extended clone flag version. I think.

> 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.

The patch looks good and I compiled it and booted on x64 and
x86_64.

I think we should add the unshare support before sending to
andrew and also add an extended flag array to show how it will
be used. I have a mq_namespace patchset pending we could use
for that and send all together ?

C.

> 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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Cedric Le Goater on Tue, 15 Jan 2008 13:52:02 GMT
View Forum Message <> Reply to Message

> @@ -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;
> + }
> +

it's probably better do to :

```
 carg = get_long_clone_arg(child_tidptr);
 if (IS_ERR(carg)) {
  retval = PTR_ERR(carg);
  goto bad_fork_cleanup_policy;
 }
```

C.

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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Daniel Hokka Zakrisso on Tue, 15 Jan 2008 14:34:41 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> 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);
```

This means that software built against a newer kernel won't work on an
older one. Surely that's not intended?

```
> + 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;
```

--
Daniel Hokka Zakrisson

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

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Cedric Le Goater on Tue, 15 Jan 2008 14:48:13 GMT
View Forum Message <> Reply to Message

Cedric Le Goater wrote:
> Pavel Emelyanov wrote:
>> 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.
>
> hmm, I wonder how lkml@ will react to this. do we have
> similar apis in the kernel ?
>
>> Sure, we lose the ability to clone tasks with extended
>> argument and the CLONE_CHILD_SETTID/CLEARTID, but do we
>> really need this?
>
> not in the extended clone flag version. I think.
>
>> 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.
>
> The patch looks good and I compiled it and booted on x64 and
> x86_64.
>
> I think we should add the unshare support before sending to
> andrew and also add an extended flag array to show how it will
> be used. I have a mq_namespace patchset pending we could use
> for that and send all together ?

Here's the unshare part if you want to fold that with your patch.

Thanks,

C.

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 include/linux/nsproxy.h  |    2 +-
 include/linux/syscalls.h |    2 +-
 kernel/fork.c            |   19 +++++++++++++++++----
 kernel/nsproxy.c         |    7 +++++---
 4 files changed, 21 insertions(+), 9 deletions(-)

Index: 2.6.24-rc5-mm1/include/linux/nsproxy.h
===================================================================
--- 2.6.24-rc5-mm1.orig/include/linux/nsproxy.h
+++ 2.6.24-rc5-mm1/include/linux/nsproxy.h
@@ -68,7 +68,7 @@ void exit_task_namespaces(struct task_st
 void switch_task_namespaces(struct task_struct *tsk, struct nsproxy *new);
 void free_nsproxy(struct nsproxy *ns);
 int unshare_nsproxy_namespaces(unsigned long, struct nsproxy **,
-       struct fs_struct *);
+               struct fs_struct *, struct long_clone_arg *carg);

 static inline void put_nsproxy(struct nsproxy *ns)
 {
Index: 2.6.24-rc5-mm1/include/linux/syscalls.h
===================================================================
--- 2.6.24-rc5-mm1.orig/include/linux/syscalls.h
+++ 2.6.24-rc5-mm1/include/linux/syscalls.h
@@ -585,7 +585,7 @@ asmlinkage long compat_sys_newfstatat(un
                                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,
Index: 2.6.24-rc5-mm1/kernel/fork.c
===================================================================
--- 2.6.24-rc5-mm1.orig/kernel/fork.c
+++ 2.6.24-rc5-mm1/kernel/fork.c
@@ -1700,7 +1700,7 @@ static int unshare_semundo(unsigned long
  * 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;
@@ -1709,6 +1709,7 @@ asmlinkage long sys_unshare(unsigned lon
        struct files_struct *fd, *new_fd = NULL;
        struct sem_undo_list *new_ulist = NULL;
        struct nsproxy *new_nsproxy = NULL;
+       struct long_clone_arg *carg = NULL;

        check_unshare_flags(&unshare_flags);

@@ -1717,11 +1718,19 @@ asmlinkage long sys_unshare(unsigned lon
        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_NEWCLONE))
                goto bad_unshare_out;

+       if (unshare_flags & CLONE_NEWCLONE) {
+               carg = get_long_clone_arg(flagptr);
+               if (IS_ERR(carg)) {
+                       err = PTR_ERR(carg);
+                       goto bad_unshare_out;
+               }
+       }
+
        if ((err = unshare_thread(unshare_flags)))
-               goto bad_unshare_out;
+               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)))
@@ -1733,7 +1742,7 @@ asmlinkage long sys_unshare(unsigned lon
```

```
        if ((err = unshare_semundo(unshare_flags, &new_ulist)))
                goto bad_unshare_cleanup_fd;
        if ((err = unshare_nsproxy_namespaces(unshare_flags, &new_nsproxy,
-                       new_fs)))
+                        new_fs, carg)))
                goto bad_unshare_cleanup_semundo;

        if (new_fs ||  new_mm || new_fd || new_ulist || new_nsproxy) {
@@ -1791,6 +1800,8 @@ bad_unshare_cleanup_fs:
                put_fs_struct(new_fs);

 bad_unshare_cleanup_thread:
+bad_unshare_cleanup_carg:
+       kfree(carg);
 bad_unshare_out:
        return err;
 }
Index: 2.6.24-rc5-mm1/kernel/nsproxy.c
===================================================================
--- 2.6.24-rc5-mm1.orig/kernel/nsproxy.c
+++ 2.6.24-rc5-mm1/kernel/nsproxy.c
@@ -182,19 +182,20 @@ void free_nsproxy(struct nsproxy *ns)
  * On success, returns the new nsproxy.
  */
 int unshare_nsproxy_namespaces(unsigned long unshare_flags,
-               struct nsproxy **new_nsp, struct fs_struct *new_fs)
+               struct nsproxy **new_nsp, struct fs_struct *new_fs,
+               struct long_clone_arg *carg)
 {
        int err = 0;

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

        if (!capable(CAP_SYS_ADMIN))
                return -EPERM;

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

_____
Containers mailing list
Containers@lists.linux-foundation.org

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Oren Laadan on Tue, 15 Jan 2008 15:17:21 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> 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.

how about "CLONE_EXTEND" ?

> 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;
> };

how about "ext_clone_arg" ?

(both suggestion make the use more explicit and are more
consistent with each other; but definitely a nit ...)

>
> 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.

Since the same kernel version (maj/min) can be compiled with
different config options, need to ensure that not only are
args added at the end, but also without #if/#ifdef around
them.

That also means, that if a feature isn't compile (but the
size argument remains larger because of fields required for
other arguments, a user program currently has no way to
figure out what's available and supported by the kernel.

(see also comment below about the code).

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

not necessarily. the relevant field can be added (even
already now) inside long_clone_arg, e.g. child_tidptr;
so if the extra bit is set, _and_ ....CLEARTID is set,
the pointer is found inside the extra struct.

in other words we don't give up the functionality.

>
> 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);

what about:
 if (size < sizeof(int))
  return ERR_PTR(-EINVAL);

I wonder how a caller could figure out what _is_ supported by the
kernel on which it is running - that is, what is the "perfect" value
for the "size" field (and whether that is necessary ?)
For instance, the kernel could "put_user(sizeof(...), child_tidptr)"
if the original size given by the caller is 0.

> +
> + 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

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Cedric Le Goater on Tue, 15 Jan 2008 15:56:39 GMT
View Forum Message <> Reply to Message

Oren Laadan wrote:
>
> Pavel Emelyanov wrote:
>> 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.
>
> how about "CLONE_EXTEND" ?
>
>> 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;
>> };
>
> how about "ext_clone_arg" ?
>
> (both suggestion make the use more explicit and are more
> consistent with each other; but definitely a nit ...)

yeah I agree. The naming can be improved but let's just wait
for the patch to be sent on lkml@. I'm sure we will have
plenty of feedback.

however, the last clone flag name should be consistent with the
structure name. CLONE_NEWCLONE is not.

C.

_____
Containers mailing list

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Dave Hansen on Tue, 15 Jan 2008 17:54:20 GMT
View Forum Message <> Reply to Message

On Tue, 2008-01-15 at 15:50 +0300, Pavel Emelyanov wrote:
> +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);

This little bit means that any newer app (with a large
long_clone_arg->size) trying to run on an older kernel (with a smaller
struct) would simply fail to run clone(). Perhaps it shouldn't be _so_
generic as to allow anything in the struct and should stick to bits.
That way, we can actually go look to see whether there are any _unknown_
bits set just like we do now with clone flags.

The more I think about this, the more nervous I get about it. It is
really neat, but has a bit of the stink of ioctl()s on it. I'd
personally rather see a new system call.

But, this seems like a good Linus question. Want to keep us on cc, but
run it by him (and the rest of LKML)?

-- Dave

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

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by serue on Tue, 15 Jan 2008 21:32:04 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> 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.

If we're going to go this route, the patch does look correct
to me.

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

Reviewed-by: Serge Hallyn <serue@us.ibm.com>

>
> ---
>
> 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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by serue on Tue, 15 Jan 2008 21:40:18 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> Cedric Le Goater wrote:
> > Pavel Emelyanov wrote:
> >> 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.
> >
> > hmm, I wonder how lkml@ will react to this. do we have
> > similar apis in the kernel ?
> >
> >> Sure, we lose the ability to clone tasks with extended
> >> argument and the CLONE_CHILD_SETTID/CLEARTID, but do we

> >> really need this?
> >
> > not in the extended clone flag version. I think.
> >
> >> 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.
> >
> > The patch looks good and I compiled it and booted on x64 and
> > x86_64.
> >
> > I think we should add the unshare support before sending to
> > andrew and also add an extended flag array to show how it will
> > be used. I have a mq_namespace patchset pending we could use
> > for that and send all together ?
>
> Here's the unshare part if you want to fold that with your patch.
>
> Thanks,
>
> C.
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
> ---
>  include/linux/nsproxy.h  |    2 +-
>  include/linux/syscalls.h |    2 +-
>  kernel/fork.c            |   19 ++++++++++++++++----
>  kernel/nsproxy.c         |    7 ++++---
>  4 files changed, 21 insertions(+), 9 deletions(-)
>
> Index: 2.6.24-rc5-mm1/include/linux/nsproxy.h
> ===================================================================
> --- 2.6.24-rc5-mm1.orig/include/linux/nsproxy.h
> +++ 2.6.24-rc5-mm1/include/linux/nsproxy.h
> @@ -68,7 +68,7 @@ void exit_task_namespaces(struct task_st
>  void switch_task_namespaces(struct task_struct *tsk, struct nsproxy *new);
>  void free_nsproxy(struct nsproxy *ns);

>  int unshare_nsproxy_namespaces(unsigned long, struct nsproxy **,
> -       struct fs_struct *);
> +               struct fs_struct *, struct long_clone_arg *carg);
>
>  static inline void put_nsproxy(struct nsproxy *ns)
>  {
> Index: 2.6.24-rc5-mm1/include/linux/syscalls.h
> ===================================================================
> --- 2.6.24-rc5-mm1.orig/include/linux/syscalls.h
> +++ 2.6.24-rc5-mm1/include/linux/syscalls.h
> @@ -585,7 +585,7 @@ asmlinkage long compat_sys_newfstatat(un
>                                       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);

Hmm, why not properly call this a struct long_clone_arg __user *flagptr
here?

>
>  asmlinkage long sys_splice(int fd_in, loff_t __user *off_in,
>                             int fd_out, loff_t __user *off_out,
> Index: 2.6.24-rc5-mm1/kernel/fork.c
> ===================================================================
> --- 2.6.24-rc5-mm1.orig/kernel/fork.c
> +++ 2.6.24-rc5-mm1/kernel/fork.c
> @@ -1700,7 +1700,7 @@ static int unshare_semundo(unsigned long
>   * 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;
> @@ -1709,6 +1709,7 @@ asmlinkage long sys_unshare(unsigned lon
>         struct files_struct *fd, *new_fd = NULL;
>         struct sem_undo_list *new_ulist = NULL;
>         struct nsproxy *new_nsproxy = NULL;
> +       struct long_clone_arg *carg = NULL;
>
>         check_unshare_flags(&unshare_flags);
>
> @@ -1717,11 +1718,19 @@ asmlinkage long sys_unshare(unsigned lon
>         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_NEWCLONE))
>                 goto bad_unshare_out;
>
> +       if (unshare_flags & CLONE_NEWCLONE) {
> +               carg = get_long_clone_arg(flagptr);
> +               if (IS_ERR(carg)) {
> +                       err = PTR_ERR(carg);
> +                       goto bad_unshare_out;
> +               }
> +       }
> +
>         if ((err = unshare_thread(unshare_flags)))
> -               goto bad_unshare_out;
> +               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)))
> @@ -1733,7 +1742,7 @@ asmlinkage long sys_unshare(unsigned lon
>         if ((err = unshare_semundo(unshare_flags, &new_ulist)))
>                 goto bad_unshare_cleanup_fd;
>         if ((err = unshare_nsproxy_namespaces(unshare_flags, &new_nsproxy,
> -                                             new_fs)))
> +                                             new_fs, carg)))
>                 goto bad_unshare_cleanup_semundo;
>
>         if (new_fs ||  new_mm || new_fd || new_ulist || new_nsproxy) {
> @@ -1791,6 +1800,8 @@ bad_unshare_cleanup_fs:
>                 put_fs_struct(new_fs);
>
>  bad_unshare_cleanup_thread:
> +bad_unshare_cleanup_carg:
> +       kfree(carg);
>  bad_unshare_out:
>         return err;
>  }
> Index: 2.6.24-rc5-mm1/kernel/nsproxy.c
> ===================================================================
> --- 2.6.24-rc5-mm1.orig/kernel/nsproxy.c
> +++ 2.6.24-rc5-mm1/kernel/nsproxy.c
> @@ -182,19 +182,20 @@ void free_nsproxy(struct nsproxy *ns)
>   * On success, returns the new nsproxy.
>   */
>  int unshare_nsproxy_namespaces(unsigned long unshare_flags,
> -               struct nsproxy **new_nsp, struct fs_struct *new_fs)
> +               struct nsproxy **new_nsp, struct fs_struct *new_fs,
> +               struct long_clone_arg *carg)
>  {
```

```
>       int err = 0;
>
>       if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
> -                            CLONE_NEWUSER | CLONE_NEWNET)))
> +                            CLONE_NEWUSER | CLONE_NEWNET | CLONE_NEWCLONE)))
>               return 0;
>
>       if (!capable(CAP_SYS_ADMIN))
>               return -EPERM;
>
>       *new_nsp = create_new_namespaces(unshare_flags, current,
> -                             new_fs ? new_fs : current->fs, NULL);
> +                             new_fs ? new_fs : current->fs, carg);
>       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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by serue on Tue, 15 Jan 2008 21:46:31 GMT
View Forum Message <> Reply to Message

Quoting Daniel Hokka Zakrisson (daniel@hozac.com):
> Pavel Emelyanov wrote:
> > 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);
>
> This means that software built against a newer kernel won't work on an
> older one. Surely that's not intended?

Well the idea would be that if userspace knows about 64 additional bits
but is only asking for bit 3, it would pass

```
{
 size=sizeof(int)+sizeof(__u32),
 u32_1 = 000...000100,
}
```

rather than

```
{
 size=sizeof(int)+2*sizeof(__u32_),
 u32_1 = 000...000100,
 u32_2 = 000...000000,
}
```

And, of course, if if it should set a bit in u32_2, then the kernel
which only knows about 32 additional bits doesn't support the requested
feature and should in fact return -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;
>
> --
> Daniel Hokka Zakrisson

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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Pavel Emelianov on Wed, 16 Jan 2008 07:26:36 GMT
View Forum Message <> Reply to Message

[snip]

>> +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);
>
> This means that software built against a newer kernel won't work on an
> older one. Surely that's not intended?

It is intended. If I ask an old kernel to clone the mq namespace, but
it doesn't support such, that I'd better like to get an -EINVAL error
rather than be silently held in an old global namespace.

[snip]

_____

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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Pavel Emelianov on Wed, 16 Jan 2008 07:37:16 GMT
View Forum Message <> Reply to Message

Oren Laadan wrote:
>
> Pavel Emelyanov wrote:
>> 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.
>
> how about "CLONE_EXTEND" ?

I don't mind. The names I selected are not perfect. Better
ones are always welcome.

>> 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;
>> };
>
> how about "ext_clone_arg" ?
>
> (both suggestion make the use more explicit and are more
> consistent with each other; but definitely a nit ...)

"Extended" sounds good, thanks.

>> 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.

>
> Since the same kernel version (maj/min) can be compiled with
> different config options, need to ensure that not only are
> args added at the end, but also without #if/#ifdef around
> them.

Sure, but since this struct is declared outside the
#ifdef __KERNEL__ then the internal ifdefs will look
strange. But I will mention this in comment.

> That also means, that if a feature isn't compile (but the
> size argument remains larger because of fields required for
> other arguments, a user program currently has no way to
> figure out what's available and supported by the kernel.
> (see also comment below about the code).
>
>> Sure, we lose the ability to clone tasks with extended
>> argument and the CLONE_CHILD_SETTID/CLEARTID, but do we
>> really need this?
>
> not necessarily. the relevant field can be added (even
> already now) inside long_clone_arg, e.g. child_tidptr;
> so if the extra bit is set, _and_ ....CLEARTID is set,
> the pointer is found inside the extra struct.
>
> in other words we don't give up the functionality.
>
>> 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);
>
> what about:
>  if (size < sizeof(int))
>   return ERR_PTR(-EINVAL);

Hm... Makes sense.

> I wonder how a caller could figure out what _is_ supported by the
> kernel on which it is running - that is, what is the "perfect" value
> for the "size" field (and whether that is necessary ?)
> For instance, the kernel could "put_user(sizeof(...), child_tidptr)"
> if the original size given by the caller is 0.

You mean use the clone to get what the kernel support? I don't
think this is good.

What would the program need to find out what the kernel support
at run-time? If we want to create some extra namespace and the
kernel returns -EINVAL then there's no need to restart the syscall
w/o asking it to clone this namespace. I think that once user
needs some namespace it finds out which kernel version he needs
and goes on.

>> +
>> + 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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Pavel Emelianov on Wed, 16 Jan 2008 07:39:56 GMT
View Forum Message <> Reply to Message

> Here's the unshare part if you want to fold that with your patch.

Thanks, Cedric. I'll pick that up.

> Thanks,

OK, thanks everyone, I'll try to cook the final patch today
and send it to Andrew as an RFC.

> C.

Thanks,

Pavel

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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Daniel Hokka Zakrisso on Wed, 16 Jan 2008 07:58:51 GMT
View Forum Message <> Reply to Message

Pavel Emelyanov wrote:
> [snip]
>
>>> +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);
>>
>> This means that software built against a newer kernel won't work on an
>> older one. Surely that's not intended?
>
> It is intended. If I ask an old kernel to clone the mq namespace, but
> it doesn't support such, that I'd better like to get an -EINVAL error
> rather than be silently held in an old global namespace.

That rules out using the struct for things like child_tidptr, the desired
pid for C/R, etc. I think the version Dave Hansen proposed would be
better, or if it's really just for bits, use an array rather than a struct
to make that obvious.

> [snip]
>

--
Daniel Hokka Zakrisson

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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Daniel Hokka Zakrisso on Wed, 16 Jan 2008 08:07:21 GMT
View Forum Message <> Reply to Message

I wrote:
> Pavel Emelyanov wrote:
>> [snip]
>>
>>>> +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);
>>>
>>> This means that software built against a newer kernel won't work on an
>>> older one. Surely that's not intended?
>>
>> It is intended. If I ask an old kernel to clone the mq namespace, but
>> it doesn't support such, that I'd better like to get an -EINVAL error
>> rather than be silently held in an old global namespace.
>
> That rules out using the struct for things like child_tidptr, the desired
> pid for C/R, etc. I think the version Dave Hansen proposed would be
> better, or if it's really just for bits, use an array rather than a struct
> to make that obvious.

s/Dave Hansen/Oren Laadan/
Sorry, too early for me.

>> [snip]
>>

--
Daniel Hokka Zakrisson

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

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Pavel Emelianov on Wed, 16 Jan 2008 08:09:06 GMT

Daniel Hokka Zakrisson wrote:
> Pavel Emelyanov wrote:
>> [snip]
>>
>>>> +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);
>>> This means that software built against a newer kernel won't work on an
>>> older one. Surely that's not intended?
>> It is intended. If I ask an old kernel to clone the mq namespace, but
>> it doesn't support such, that I'd better like to get an -EINVAL error
>> rather than be silently held in an old global namespace.
>
> That rules out using the struct for things like child_tidptr, the desired
> pid for C/R, etc. I think the version Dave Hansen proposed would be

A new system call? Ok - what arguments should it take?

> better, or if it's really just for bits, use an array rather than a struct
> to make that obvious.

It's for anything you'd like to tell to clone(). In the nearest future - for
more bits to clone new namespaces. In the far future - for pid to create the
task with (for c/r jobs). In the the far-far future - for anything you will
need then.

>> [snip]
>>
>

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

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Cedric Le Goater on Wed, 16 Jan 2008 08:15:13 GMT

Pavel Emelyanov wrote:
>> Here's the unshare part if you want to fold that with your patch.
>
> Thanks, Cedric. I'll pick that up.
>
>> Thanks,
>
> OK, thanks everyone, I'll try to cook the final patch today
> and send it to Andrew as an RFC.

yes.

I'd changed the subject of the patch to something more explicit
on what we're doing :

'use child_tidptr to extend clone() flags'

we need feedback from the community on this clone extension ! :)

C.

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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Dave Hansen on Wed, 16 Jan 2008 17:52:57 GMT


On Wed, 2008-01-16 at 10:26 +0300, Pavel Emelyanov wrote:
> > This means that software built against a newer kernel won't work on an
> > older one. Surely that's not intended?
>
> It is intended. If I ask an old kernel to clone the mq namespace, but
> it doesn't support such, that I'd better like to get an -EINVAL error
> rather than be silently held in an old global namespace.

There's at least one thing that this doesn't handle. Let's say we have
a clone64(long, long) but that the kernel we're running is old and don't
know about any valid flags in the second long, yet.

Somebody who knows about lots of new flags 10 years down the line passes
clone(0x1234, 0x0) to the old kernel. That's OK, because the kernel
knows about all of the flags that *ARE* set.

Your proposal is basically this:

clone(flags, {size, stuff});

and if the in-kernel size isn't exactly the same as the size passed in,
it punts.  It doesn't matter if anything in "stuff" was actually going
to be used, or if the old kernel *could* have handled it, it just punts.

Honestly, I don't think it's that big of a deal, I just wish you'd
acknowledge it. ;)

-- Dave

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

---

Subject: Re: [PATCH] An attempt to have an unlimitedly extendable sys_clone
Posted by Cedric Le Goater on Mon, 21 Jan 2008 10:36:38 GMT
View Forum Message <> Reply to Message

Hello Pavel !

Pavel Emelyanov wrote:
>> Here's the unshare part if you want to fold that with your patch.
>
> Thanks, Cedric. I'll pick that up.
>
>> Thanks,
>
> OK, thanks everyone, I'll try to cook the final patch today
> and send it to Andrew as an RFC.

As Al said it, it's probably not the direction to follow to extend the
clone flags :) But the problem being real, I think we need to insist.

First, we could simplify the API by using child_tidptr to pass a 64bits
array of flags. This would still be a brutal syscall argument hijack but
we can't do better without a new syscall.

Next, we could send the same code using a new syscall.

I have time to work on that.

C.

_____

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