Subject: [PATCH 1/2] pidns: Don't allow new pids after the namespace is dead.
Posted by Daniel Lezcano on Tue, 15 Feb 2011 16:53:43 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

In the case of unsharing or joining a pid namespace, it becomes
possible to attempt to allocate a pid after zap_pid_namespace has
killed everything in the namespace. Close the hole for now by simply
not allowing any of those pid allocations to succeed. At least for
now it is too strange to think about.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Signed-off-by: Daniel Lezcano <daniel.lezcano@free.fr>
---
include/linux/pid_namespace.h | 1 +
kernel/pid.c | 4 ++++
kernel/pid_namespace.c | 2 ++
3 files changed, 7 insertions(+), 0 deletions(-)

```
diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index 38d1032..b447d37 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -20,6 +20,7 @@ struct pid_namespace {
  struct kref kref;
  struct pidmap pidmap[PIDMAP_ENTRIES];
  int last_pid;
+ atomic_t dead;
  struct task_struct *child_reaper;
  struct kmem_cache *pid_cachep;
  unsigned int level;
diff --git a/kernel/pid.c b/kernel/pid.c
index 39b65b6..e996950 100644
--- a/kernel/pid.c
+++ b/kernel/pid.c
@@ -282,6 +282,10 @@ struct pid *alloc_pid(struct pid_namespace *ns)
  struct pid_namespace *tmp;
  struct upid *upid;

+ pid = NULL;
+ if (atomic_read(&ns->dead))
+  goto out;
+
  pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
  if (!pid)
   goto out;
diff --git a/kernel/pid_namespace.c b/kernel/pid_namespace.c
```

```
index e9c9adc..e8ea25d 100644
--- a/kernel/pid_namespace.c
+++ b/kernel/pid_namespace.c
@@ -90,6 +90,7 @@ static struct pid_namespace *create_pid_namespace(struct
pid_namespace *parent_p
  kref_init(&ns->kref);
  ns->level = level;
  ns->parent = get_pid_ns(parent_pid_ns);
+ atomic_set(&ns->dead, 0);

  set_bit(0, ns->pidmap[0].page);
  atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
@@ -164,6 +165,7 @@ void zap_pid_ns_processes(struct pid_namespace *pid_ns)
   *
   */
  read_lock(&tasklist_lock);
+ atomic_set(&pid_ns->dead, 1);
  nr = next_pidmap(pid_ns, 1);
  while (nr > 0) {
   rcu_read_lock();
--
1.7.1
```

Subject: [PATCH 2/2] pidns: Support unsharing the pid namespace.
Posted by Daniel Lezcano on Tue, 15 Feb 2011 16:53:44 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com>

- Allow CLONEW_NEWPID into unshare.
- Pass both nsproxy->pid_ns and task_active_pid_ns to copy_pid_ns
  As they can now be different.

Unsharing of the pid namespace unlike unsharing of other namespaces
does not take effect immediately. Instead it affects the children
created with fork and clone. The first of these children becomes the init
process of the new pid namespace, the rest become oddball children
of pid 0. From the point of view of the new pid namespace the process
that created it is pid 0, as it's pid does not map.

A couple of different semantics were considered but this one was
settled on because it is easy to implement and it is usable from

pam modules. The core reasons for the existence of unshare.

I took a survey of the callers of pam modules and the following
appears to be a representative sample of their logic.

```
{
 setup stuff include pam
 child = fork();
 if (!child) {
  setuid()
                exec /bin/bash
        }
        waitpid(child);

        pam and other cleanup
}
```

As you can see there is a fork to create the unprivileged user space process. Which means that the unprivileged user space process will appear as pid 1 in the new pid namespace. Further most login processes do not cope with extraneous children which means shifting the duty of reaping extraneous child process to the creator of those extraneous children makes the system more comprehensible.

The practical reason for this set of pid namespace semantics is that it is simple to implement and verify they work correctly. Whereas an implementation that requires changing the struct pid on a process comes with a lot more races and pain. Not the least of which is that glibc caches getpid().

These semantics are implemented by having two notions of the pid namespace of a process. There is task_active_pid_ns which is the pid namspace the process was created with and the pid namespace that all pids are presented to that process in. The task_active_pid_ns is stored in the struct pid of the task.

There is the pid namespace that will be used for children that pid namespace is stored in task->nsproxy->pid_ns.

There is one really nasty corner case in all of this. Which pid namespace are you in if your parent unshared it's pid namespace and then on clone you also unshare the pid namespace. To me there are only two possible answers. Either the cases is so bizarre and we deny it completely. or the new pid namespace is a descendent of our parent's active pid namespace, and we ignore the task->nsproxy->pid_ns.

To that end I have modified copy_pid_ns to take both of these
pid namespaces. The active pid namespace and the default
pid namespace of children. Allowing me to simply implement
unsharing a pid namespace in clone after already unsharing
a pid namespace with unshare.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
Signed-off-by: Daniel Lezcano <daniel.lezcano@free.fr>
---
 include/linux/pid_namespace.h | 14 ++++++++-----
 kernel/fork.c | 3 ++-
 kernel/nsproxy.c | 5 +++--
 kernel/pid_namespace.c | 8 +++++---
 4 files changed, 19 insertions(+), 11 deletions(-)

diff --git a/include/linux/pid_namespace.h b/include/linux/pid_namespace.h
index b447d37..4316347 100644
--- a/include/linux/pid_namespace.h
+++ b/include/linux/pid_namespace.h
@@ -43,7 +43,10 @@ static inline struct pid_namespace *get_pid_ns(struct pid_namespace *ns)
  return ns;
 }

-extern struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *ns);
+extern struct pid_namespace *copy_pid_ns(unsigned long flags,
+	struct pid_namespace *default_ns,
+	struct pid_namespace *active_ns);
+
 extern void free_pid_ns(struct kref *kref);
 extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

@@ -61,12 +64,13 @@ static inline struct pid_namespace *get_pid_ns(struct pid_namespace
*ns)
  return ns;
 }

-static inline struct pid_namespace *
-copy_pid_ns(unsigned long flags, struct pid_namespace *ns)
+static inline struct pid_namespace *copy_pid_ns(unsigned long flags,
+	struct pid_namespace *default_ns,
+	struct pid_namespace *active_ns)
 {
  if (flags & CLONE_NEWPID)
- ns = ERR_PTR(-EINVAL);
- return ns;
+ return ERR_PTR(-EINVAL);
+ return default_ns;
 }

```
 static inline void put_pid_ns(struct pid_namespace *ns)
diff --git a/kernel/fork.c b/kernel/fork.c
index e7a5907..4b019f1 100644
--- a/kernel/fork.c
+++ b/kernel/fork.c
@@ -1633,7 +1633,8 @@ SYSCALL_DEFINE1(unshare, unsigned long, unshare_flags)
  err = -EINVAL;
  if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
-    CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWNET))
+          CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWNET|
+          CLONE_NEWPID))
   goto bad_unshare_out;

  /*
diff --git a/kernel/nsproxy.c b/kernel/nsproxy.c
index f74e6c0..a9cf251 100644
--- a/kernel/nsproxy.c
+++ b/kernel/nsproxy.c
@@ -81,7 +81,8 @@ static struct nsproxy *create_new_namespaces(unsigned long flags,
   goto out_ipc;
  }

- new_nsp->pid_ns = copy_pid_ns(flags, task_active_pid_ns(tsk));
+ new_nsp->pid_ns = copy_pid_ns(flags, tsk->nsproxy->pid_ns,
+          task_active_pid_ns(tsk));
  if (IS_ERR(new_nsp->pid_ns)) {
   err = PTR_ERR(new_nsp->pid_ns);
   goto out_pid;
@@ -185,7 +186,7 @@ int unshare_nsproxy_namespaces(unsigned long unshare_flags,
  int err = 0;

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

  if (!capable(CAP_SYS_ADMIN))
diff --git a/kernel/pid_namespace.c b/kernel/pid_namespace.c
index e8ea25d..9e101c1 100644
--- a/kernel/pid_namespace.c
+++ b/kernel/pid_namespace.c
@@ -123,13 +123,15 @@ static void destroy_pid_namespace(struct pid_namespace *ns)
  kmem_cache_free(pid_ns_cachep, ns);
 }

-struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
```

```
+struct pid_namespace *copy_pid_ns(unsigned long flags,
+       struct pid_namespace *default_ns,
+       struct pid_namespace *active_ns)
 {
  if (!(flags & CLONE_NEWPID))
-  return get_pid_ns(old_ns);
+  return get_pid_ns(default_ns);
  if (flags & (CLONE_THREAD|CLONE_PARENT))
   return ERR_PTR(-EINVAL);
- return create_pid_namespace(old_ns);
+ return create_pid_namespace(active_ns);
 }

 void free_pid_ns(struct kref *kref)
--
1.7.1
```

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

---

Subject: Re: [PATCH 1/2] pidns: Don't allow new pids after the namespace is dead.
Posted by Oleg Nesterov on Tue, 15 Feb 2011 18:30:28 GMT
View Forum Message <> Reply to Message

On 02/15, Daniel Lezcano wrote:
>
> In the case of unsharing or joining a pid namespace, it becomes
> possible to attempt to allocate a pid after zap_pid_namespace has
> killed everything in the namespace. Close the hole for now by simply
> not allowing any of those pid allocations to succeed.

Daniel, please explain more. It seems, a long ago I knew the reason
for this patch, but now I can't recall and can't understand this change.

> --- a/include/linux/pid_namespace.h
> +++ b/include/linux/pid_namespace.h
> @@ -20,6 +20,7 @@ struct pid_namespace {
>   struct kref kref;
>   struct pidmap pidmap[PIDMAP_ENTRIES];
>   int last_pid;
> + atomic_t dead;

Why atomic_t? It is used as a plain boolean.

And I can't unde

> --- a/kernel/pid.c
> +++ b/kernel/pid.c
> @@ -282,6 +282,10 @@ struct pid *alloc_pid(struct pid_namespace *ns)
> struct pid_namespace *tmp;
> struct upid *upid;
>
> + pid = NULL;
> + if (atomic_read(&ns->dead))
> + goto out;
> +

So why this is needed?

If we see ns->dead != 0 we are already killed by zap_pid_ns_processes()
which sets ns->dead = 1.

Oleg.

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

---

Subject: Re: [PATCH 2/2] pidns: Support unsharing the pid namespace.
Posted by Oleg Nesterov on Tue, 15 Feb 2011 19:01:18 GMT
View Forum Message <> Reply to Message

On 02/15, Daniel Lezcano wrote:
>
> - Pass both nsproxy->pid_ns and task_active_pid_ns to copy_pid_ns
> As they can now be different.

But since they can be different we have to convert some users of
current->nsproxy first? But that patch was dropped.

> Unsharing of the pid namespace unlike unsharing of other namespaces
> does not take effect immediately. Instead it affects the children
> created with fork and clone.

IOW, unshare(CLONE_NEWPID) implicitly affects the subsequent fork(),
using the very subtle way.

I have to admit, I can't say I like this very much. OK, if we need
this, can't we just put something into, say, signal->flags so that
copy_process can check and create the new namespace.

Also. I remember, I already saw something like this and google found my questions. I didn't actually read the new version, perhaps my concerns were already answered...

But what if the task T does unshare(CLONE_NEWPID) and then, say, pthread_create() ? Unless I missed something, the new thread won't be able to see T ?

and, in this case the exiting sub-namespace init also kills its parent?

OK, suppose it does fork() after unshare(), then another fork(). In this case the second child lives in the same namespace with init created by the 1st fork, but it is not descendant ? This means in particular that if the new init exits, zap_pid_ns_processes()-> do_wait() can't work.

Or not?

Oleg.

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

---

Subject: [PATCH 0/1] Was: pidns: Support unsharing the pid namespace.
Posted by Oleg Nesterov on Tue, 15 Feb 2011 19:15:21 GMT
View Forum Message <> Reply to Message

On 02/15, Oleg Nesterov wrote:
>
> On 02/15, Daniel Lezcano wrote:
> >
> > - Pass both nsproxy->pid_ns and task_active_pid_ns to copy_pid_ns
> >   As they can now be different.
>
> But since they can be different we have to convert some users of
> current->nsproxy first? But that patch was dropped.
>
> > Unsharing of the pid namespace unlike unsharing of other namespaces
> > does not take effect immediately. Instead it affects the children
> > created with fork and clone.
>
> IOW, unshare(CLONE_NEWPID) implicitly affects the subsequent fork(),
> using the very subtle way.
>

> I have to admit, I can't say I like this very much. OK, if we need
> this, can't we just put something into, say, signal->flags so that
> copy_process can check and create the new namespace.
>
> Also. I remember, I already saw something like this and google found
> my questions. I didn't actually read the new version, perhaps my
> concerns were already answered...
>
> But what if the task T does unshare(CLONE_NEWPID) and then, say,
> pthread_create() ? Unless I missed something, the new thread won't
> be able to see T ?
>
> and, in this case the exiting sub-namespace init also kills its
> parent?
>
> OK, suppose it does fork() after unshare(), then another fork().
> In this case the second child lives in the same namespace with
> init created by the 1st fork, but it is not descendant ? This means
> in particular that if the new init exits, zap_pid_ns_processes()->
> do_wait() can't work.
>
> Or not?

And, can't resist. If we are going to change sys_unshare(), I'd like
very much to cleanup it first.

Dear all! I promise, I will resend this patch forever until somebody
explains me why it is constantly ignored ;)

Oleg.

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

---

Subject: [PATCH 1/1][3rd resend] sys_unshare: remove the dead CLONE_THREAD/SIGHAND/VM code
Posted by Oleg Nesterov on Tue, 15 Feb 2011 19:17:11 GMT
View Forum Message <> Reply to Message

Cleanup: kill the dead code which does nothing but complicates the code
and confuses the reader.

sys_unshare(CLONE_THREAD/SIGHAND/VM) is not really implemented, and I doubt
very much it will ever work. At least, nobody even tried since the original
"unshare system call -v5: system call handler function" commit

99d1419d96d7df9cfa56bc977810be831bd5ef64 was applied more than 4 years ago.

And the code is not consistent. unshare_thread() always fails unconditionally,
while unshare_sighand() and unshare_vm() pretend to work if there is nothing
to unshare.

Remove unshare_thread(), unshare_sighand(), unshare_vm() helpers and related
variables and add a simple CLONE_THREAD | CLONE_SIGHAND| CLONE_VM check into
check_unshare_flags().

Also, move the "CLONE_NEWNS needs CLONE_FS" check from check_unshare_flags()
to sys_unshare(). This looks more consistent and matches the similar
do_sysvsem check in sys_unshare().

Note: with or without this patch "atomic_read(mm->mm_users) > 1" can give
a false positive due to get_task_mm().

Signed-off-by: Oleg Nesterov <oleg@redhat.com>
Acked-by: Roland McGrath <roland@redhat.com>
---

 kernel/fork.c | 123 ++++++++++++--
 1 file changed, 25 insertions(+), 98 deletions(-)

--- 2.6.37/kernel/fork.c~unshare-killcrap 2010-11-05 18:03:28.000000000 +0100
+++ 2.6.37/kernel/fork.c 2010-11-05 18:09:52.000000000 +0100
@@ -1522,38 +1522,24 @@ void __init proc_caches_init(void)
 }

 /*
- * Check constraints on flags passed to the unshare system call and
- * force unsharing of additional process context as appropriate.
+ * Check constraints on flags passed to the unshare system call.
  */
-static void check_unshare_flags(unsigned long *flags_ptr)
+static int check_unshare_flags(unsigned long unshare_flags)
 {
+ if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
+   CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
+   CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWNET))
+ return -EINVAL;
  /*
- * If unsharing a thread from a thread group, must also
- * unshare vm.
- */
- if (*flags_ptr & CLONE_THREAD)
- *flags_ptr |= CLONE_VM;
-

```
- /*
-  * If unsharing vm, must also unshare signal handlers.
-  */
- if (*flags_ptr & CLONE_VM)
-  *flags_ptr |= CLONE_SIGHAND;
-
- /*
-  * If unsharing namespace, must also unshare filesystem information.
+  * Not implemented, but pretend it works if there is nothing to
+  * unshare. Note that unsharing CLONE_THREAD or CLONE_SIGHAND
+  * needs to unshare vm.
   */
- if (*flags_ptr & CLONE_NEWNS)
-  *flags_ptr |= CLONE_FS;
-}
-
-/*
- * Unsharing of tasks created with CLONE_THREAD is not supported yet
- */
-static int unshare_thread(unsigned long unshare_flags)
-{
- if (unshare_flags & CLONE_THREAD)
-  return -EINVAL;
+ if (unshare_flags & (CLONE_THREAD | CLONE_SIGHAND | CLONE_VM)) {
+  /* FIXME: get_task_mm() increments ->mm_users */
+  if (atomic_read(&current->mm->mm_users) > 1)
+   return -EINVAL;
+ }

  return 0;
 }
@@ -1580,34 +1566,6 @@ static int unshare_fs(unsigned long unsh
 }

 /*
- * Unsharing of sighand is not supported yet
- */
-static int unshare_sighand(unsigned long unshare_flags, struct sighand_struct **new_sighp)
-{
- struct sighand_struct *sigh = current->sighand;
-
- if ((unshare_flags & CLONE_SIGHAND) && atomic_read(&sigh->count) > 1)
-  return -EINVAL;
- else
-  return 0;
-}
-
-/*
```

```
- * Unshare vm if it is being shared
- */
-static int unshare_vm(unsigned long unshare_flags, struct mm_struct **new_mmp)
-{
- struct mm_struct *mm = current->mm;
-
- if ((unshare_flags & CLONE_VM) &&
-     (mm && atomic_read(&mm->mm_users) > 1)) {
-  return -EINVAL;
- }
-
- return 0;
-}
-
-/*
  * Unshare file descriptor table if it is being shared
  */
 static int unshare_fd(unsigned long unshare_flags, struct files_struct **new_fdp)
@@ -1635,45 +1593,37 @@ static int unshare_fd(unsigned long unsh
  */
 SYSCALL_DEFINE1(unshare, unsigned long, unshare_flags)
 {
- int err = 0;
  struct fs_struct *fs, *new_fs = NULL;
- struct sighand_struct *new_sigh = NULL;
- struct mm_struct *mm, *new_mm = NULL, *active_mm = NULL;
  struct files_struct *fd, *new_fd = NULL;
  struct nsproxy *new_nsproxy = NULL;
  int do_sysvsem = 0;
+ int err;

- check_unshare_flags(&unshare_flags);
-
- /* Return -EINVAL for all unsupported flags */
- err = -EINVAL;
- if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
-     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
-     CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWNET))
+ err = check_unshare_flags(unshare_flags);
+ if (err)
   goto bad_unshare_out;

  /*
+  * If unsharing namespace, must also unshare filesystem information.
+  */
+ if (unshare_flags & CLONE_NEWNS)
+  unshare_flags |= CLONE_FS;
+ /*
```

```
   * CLONE_NEWIPC must also detach from the undolist: after switching
   * to a new ipc namespace, the semaphore arrays from the old
   * namespace are unreachable.
   */
  if (unshare_flags & (CLONE_NEWIPC|CLONE_SYSVSEM))
   do_sysvsem = 1;
- if ((err = unshare_thread(unshare_flags)))
-  goto bad_unshare_out;
  if ((err = unshare_fs(unshare_flags, &new_fs)))
-  goto bad_unshare_cleanup_thread;
- if ((err = unshare_sighand(unshare_flags, &new_sigh)))
-  goto bad_unshare_cleanup_fs;
- if ((err = unshare_vm(unshare_flags, &new_mm)))
-  goto bad_unshare_cleanup_sigh;
+  goto bad_unshare_out;
  if ((err = unshare_fd(unshare_flags, &new_fd)))
-  goto bad_unshare_cleanup_vm;
+  goto bad_unshare_cleanup_fs;
  if ((err = unshare_nsproxy_namespaces(unshare_flags, &new_nsproxy,
    new_fs)))
   goto bad_unshare_cleanup_fd;

- if (new_fs ||  new_mm || new_fd || do_sysvsem || new_nsproxy) {
+ if (new_fs || new_fd || do_sysvsem || new_nsproxy) {
   if (do_sysvsem) {
    /*
     * CLONE_SYSVSEM is equivalent to sys_exit().
@@ -1699,19 +1649,6 @@ SYSCALL_DEFINE1(unshare, unsigned long,
    spin_unlock(&fs->lock);
   }

-  if (new_mm) {
-   mm = current->mm;
-   active_mm = current->active_mm;
-   current->mm = new_mm;
-   current->active_mm = new_mm;
-   if (current->signal->oom_score_adj == OOM_SCORE_ADJ_MIN) {
-    atomic_dec(&mm->oom_disable_count);
-    atomic_inc(&new_mm->oom_disable_count);
-   }
-   activate_mm(active_mm, new_mm);
-   new_mm = mm;
-  }
-
   if (new_fd) {
    fd = current->files;
    current->files = new_fd;
@@ -1728,20 +1665,10 @@ bad_unshare_cleanup_fd:
```

```
  if (new_fd)
   put_files_struct(new_fd);

-bad_unshare_cleanup_vm:
- if (new_mm)
-  mmput(new_mm);
-
-bad_unshare_cleanup_sigh:
- if (new_sigh)
-  if (atomic_dec_and_test(&new_sigh->count))
-   kmem_cache_free(sighand_cachep, new_sigh);
-
 bad_unshare_cleanup_fs:
  if (new_fs)
   free_fs_struct(new_fs);

-bad_unshare_cleanup_thread:
 bad_unshare_out:
  return err;
 }
```

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

---

Subject: Re: [PATCH 1/2] pidns: Don't allow new pids after the namespace is dead.
Posted by Daniel Lezcano on Wed, 16 Feb 2011 23:21:12 GMT
View Forum Message <> Reply to Message

On 02/15/2011 07:30 PM, Oleg Nesterov wrote:
> On 02/15, Daniel Lezcano wrote:
>> In the case of unsharing or joining a pid namespace, it becomes
>> possible to attempt to allocate a pid after zap_pid_namespace has
>> killed everything in the namespace. Close the hole for now by simply
>> not allowing any of those pid allocations to succeed.
> Daniel, please explain more. It seems, a long ago I knew the reason
> for this patch, but now I can't recall and can't understand this change.

The idea behind unsharing the pid namespace is the current pid is not mapped in the newly created pid namespace and appears as the pid 0. When it forks, the child process becomes the init pid of the new pid namespace. When this pid namespace dies because the init pid exited, the parent process (aka pid 0) can no longer fork because the pid namespace is flagged dead. This is what does this patch.

The next patch allows a single process to spawn different processes in

different pid namespace. You can argue we can already do that with clone(CLONE_NEWPID). That's true. But if we are able to unshare the pid namespace, then the next patchset (which will come right after this one) will allow to attach a process to a namespace and the implementation will be very simple and consistent with attaching to any namespace.

>> --- a/include/linux/pid_namespace.h
>> +++ b/include/linux/pid_namespace.h
>> @@ -20,6 +20,7 @@ struct pid_namespace {
>> struct kref kref;
>> struct pidmap pidmap[PIDMAP_ENTRIES];
>> int last_pid;
>> + atomic_t dead;
> Why atomic_t? It is used as a plain boolean.
>
> And I can't unde

I think Eric used an atomic because it is lockless with alloc_pid vs zap_pid_ns_processes.

>> --- a/kernel/pid.c
>> +++ b/kernel/pid.c
>> @@ -282,6 +282,10 @@ struct pid *alloc_pid(struct pid_namespace *ns)
>> struct pid_namespace *tmp;
>> struct upid *upid;
>>
>> + pid = NULL;
>> + if (atomic_read(&ns->dead))
>> + goto out;
>> +
> So why this is needed?
>
> If we see ns->dead != 0 we are already killed by zap_pid_ns_processes()
> which sets ns->dead = 1.

The current process unshares the pid namespace.
When it forks, the child process is the pid 1. When this one exits, the zap_pid_ns_processes is called and tag the pid namespace as dead. The current process can no longer fork.

-- Daniel

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

Subject: Re: [PATCH 2/2] pidns: Support unsharing the pid namespace.
Posted by Daniel Lezcano on Wed, 16 Feb 2011 23:47:37 GMT
View Forum Message <> Reply to Message

On 02/15/2011 08:01 PM, Oleg Nesterov wrote:
> On 02/15, Daniel Lezcano wrote:
>> - Pass both nsproxy->pid_ns and task_active_pid_ns to copy_pid_ns
>> As they can now be different.
> But since they can be different we have to convert some users of
> current->nsproxy first? But that patch was dropped.
>
>> Unsharing of the pid namespace unlike unsharing of other namespaces
>> does not take effect immediately. Instead it affects the children
>> created with fork and clone.
> IOW, unshare(CLONE_NEWPID) implicitly affects the subsequent fork(),
> using the very subtle way.
>
> I have to admit, I can't say I like this very much. OK, if we need
> this, can't we just put something into, say, signal->flags so that
> copy_process can check and create the new namespace.
>
> Also. I remember, I already saw something like this and google found
> my questions. I didn't actually read the new version, perhaps my
> concerns were already answered...
>
> But what if the task T does unshare(CLONE_NEWPID) and then, say,
> pthread_create() ? Unless I missed something, the new thread won't
> be able to see T ?

Right. Is it really a problem ? I mean it is a weird use case where we
fall in a weird situation.
I suppose we can do the same weird combination with clone.
IMHO, the userspace is responsible of how it uses the syscalls. Until
the system is safe, everything is ok, no ?

> and, in this case the exiting sub-namespace init also kills its
> parent?

I don't think so because the zap_pid_ns_processes does not hit the
parent process when it browses the pidmap.

I tried the following program without problem:

```
#include <stdio.h>
#define _GNU_SOURCE
#include <sched.h>
#include <pthread.h>

void *routine(void *data)
```

```
{
        printf("pid %d!\n", getpid());
        return NULL;
}

int main(int argc, char *argv[])
{
        char **aux = &argv[1];
        pthread_t t;

        if (unshare(CLONE_NEWPID)) {
                perror("unshare");
                return -1;
        }

        if (pthread_create(&t, NULL, routine, NULL)) {
                perror("pthread_create");
                return -1;
        }

        if (pthread_join(t, NULL)) {
                perror("pthread_join");
                return -1;
        }

        printf("joined\n");

        return 0;
}
```

> OK, suppose it does fork() after unshare(), then another fork().
> In this case the second child lives in the same namespace with
> init created by the 1st fork, but it is not descendant ? This means
> in particular that if the new init exits, zap_pid_ns_processes()->
> do_wait() can't work.

Hmm, good question. IMO, we should prevent such case for now in the same
way we added the flag 'dead', IOW adding a flag 'busy' for example.

Subject: Re: [PATCH 2/2] pidns: Support unsharing the pid namespace.
Posted by Oleg Nesterov on Thu, 17 Feb 2011 20:29:59 GMT

On 02/17, Daniel Lezcano wrote:
>
> On 02/15/2011 08:01 PM, Oleg Nesterov wrote:
>>
>> I have to admit, I can't say I like this very much. OK, if we need
>> this, can't we just put something into, say, signal->flags so that
>> copy_process can check and create the new namespace.
>>
>> Also. I remember, I already saw something like this and google found
>> my questions. I didn't actually read the new version, perhaps my
>> concerns were already answered...
>>
>> But what if the task T does unshare(CLONE_NEWPID) and then, say,
>> pthread_create() ? Unless I missed something, the new thread won't
>> be able to see T ?
>
> Right. Is it really a problem ? I mean it is a weird use case where we
> fall in a weird situation.

But this is really weird! How it is possible that the parent can't see
its own child? No matter which thread did fork(), the new process is
the child of any sub-thread. More precisely, it is the child of thread
group.

> I suppose we can do the same weird combination with clone.

No, or we have the bug. If nothing else, kill() or wait() should work
equally for any sub-thread. (OK, __WNOTHREAD hack is the only exception).

>> and, in this case the exiting sub-namespace init also kills its
>> parent?
>
> I don't think so because the zap_pid_ns_processes does not hit the
> parent process when it browses the pidmap.

OK... Honestly, right now I can't understand my own question, it was
written a long ago. Probably I missed something.... but I'll recheck ;)

>> OK, suppose it does fork() after unshare(), then another fork().
>> In this case the second child lives in the same namespace with
>> init created by the 1st fork, but it is not descendant ? This means
>> in particular that if the new init exits, zap_pid_ns_processes()->
>> do_wait() can't work.
>
> Hmm, good question. IMO, we should prevent such case for now in the same
> way we added the flag 'dead', IOW adding a flag 'busy' for example.

I dunno.

As I said, I do not like this approach at all. But please feel free to
ignore, it is very easy to blaim somebody else's code without suggesting
the alternative ;)

Oleg.

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

---

Subject: Re: [PATCH 1/2] pidns: Don't allow new pids after the namespace is dead.
Posted by Oleg Nesterov on Thu, 17 Feb 2011 20:54:58 GMT
View Forum Message <> Reply to Message

On 02/17, Daniel Lezcano wrote:
>
> On 02/15/2011 07:30 PM, Oleg Nesterov wrote:
>> On 02/15, Daniel Lezcano wrote:
>>> In the case of unsharing or joining a pid namespace, it becomes
>>> possible to attempt to allocate a pid after zap_pid_namespace has
>>> killed everything in the namespace. Close the hole for now by simply
>>> not allowing any of those pid allocations to succeed.
>> Daniel, please explain more. It seems, a long ago I knew the reason
>> for this patch, but now I can't recall and can't understand this change.
>
> The idea behind unsharing the pid namespace is the current pid is not
> mapped in the newly created pid namespace and appears as the pid 0.

Well, not exactly afaics... but doesn't matter.

> When
> it forks, the child process becomes the init pid of the new pid
> namespace.

Yes, I see. And this is what I personally dislike. Because, iow,
unshare(PID) changes current->nspory->pid_ns to affect the behaviour
of copy_process() , this really looks like "action at a distance" to
me. Too subtle and fragile. But, once again, this is just imho, feel
free to ignore.

> When this pid namespace dies because the init pid exited, the
> parent process (aka pid 0) can no longer fork because the pid namespace
> is flagged dead. This is what does this patch.

OK, thanks. I seem to understand. May be ;)

I'd suggest you to add this explanation to the changelog.

>>> --- a/include/linux/pid_namespace.h
>>> +++ b/include/linux/pid_namespace.h
>>> @@ -20,6 +20,7 @@ struct pid_namespace {
>>> struct kref kref;
>>> struct pidmap pidmap[PIDMAP_ENTRIES];
>>> int last_pid;
>>> + atomic_t dead;
>> Why atomic_t? It is used as a plain boolean.
>>
>> And I can't unde
>
> I think Eric used an atomic because it is lockless with alloc_pid vs
> zap_pid_ns_processes.

Can't understand...

But anyway, I strongly believe atomic_t buys nothing in this patch.
May be it is needed for the next changes, I dunno.

Oleg.

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

---

Subject: Re: [PATCH 2/2] pidns: Support unsharing the pid namespace.
Posted by Greg Kurz on Thu, 17 Feb 2011 22:35:59 GMT
View Forum Message <> Reply to Message

On 02/17/2011 09:29 PM, Oleg Nesterov wrote:
> On 02/17, Daniel Lezcano wrote:
>>
>> On 02/15/2011 08:01 PM, Oleg Nesterov wrote:
>>>
>>> I have to admit, I can't say I like this very much. OK, if we need
>>> this, can't we just put something into, say, signal->flags so that
>>> copy_process can check and create the new namespace.
>>>
>>> Also. I remember, I already saw something like this and google found
>>> my questions. I didn't actually read the new version, perhaps my
>>> concerns were already answered...
>>>

>>> But what if the task T does unshare(CLONE_NEWPID) and then, say,
>>> pthread_create() ? Unless I missed something, the new thread won't
>>> be able to see T ?
>>
>> Right. Is it really a problem ? I mean it is a weird use case where we
>> fall in a weird situation.
>
> But this is really weird! How it is possible that the parent can't see
> its own child? No matter which thread did fork(), the new process is

Hmmm... I guess you mean the opposite. The way pid namespaces are
nested, parents always see their children. But indeed, the child thread
can't see its group leader and that's kind of unusual. Unshare a pid
namespace at your own risk. :)

> the child of any sub-thread. More precisely, it is the child of thread
> group.
>
>> I suppose we can do the same weird combination with clone.
>
> No, or we have the bug. If nothing else, kill() or wait() should work
> equally for any sub-thread. (OK, __WNOTHREAD hack is the only exception).
>
>>> and, in this case the exiting sub-namespace init also kills its
>>> parent?
>>
>> I don't think so because the zap_pid_ns_processes does not hit the
>> parent process when it browses the pidmap.
>
> OK... Honestly, right now I can't understand my own question, it was
> written a long ago. Probably I missed something.... but I'll recheck ;)
>
>>> OK, suppose it does fork() after unshare(), then another fork().
>>> In this case the second child lives in the same namespace with
>>> init created by the 1st fork, but it is not descendant ? This means
>>> in particular that if the new init exits, zap_pid_ns_processes()->
>>> do_wait() can't work.
>>
>> Hmm, good question. IMO, we should prevent such case for now in the same
>> way we added the flag 'dead', IOW adding a flag 'busy' for example.
>
> I dunno.
>
> As I said, I do not like this approach at all. But please feel free to
> ignore, it is very easy to blaim somebody else's code without suggesting
> the alternative ;)
>
> Oleg.

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

--
Gregory Kurz gkurz@fr.ibm.com
Software Engineer @ IBM/Meiosys http://www.ibm.com
Tel +33 (0)534 638 479 Fax +33 (0)561 400 420

"Anarchy is about taking complete responsibility for yourself."
Alan Moore.

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

---

Subject: Re: [PATCH 2/2] pidns: Support unsharing the pid namespace.
Posted by Oleg Nesterov on Fri, 18 Feb 2011 14:40:19 GMT
View Forum Message <> Reply to Message

On 02/17, Greg Kurz wrote:
>
> On 02/17/2011 09:29 PM, Oleg Nesterov wrote:
>> On 02/17, Daniel Lezcano wrote:
>>>
>>> On 02/15/2011 08:01 PM, Oleg Nesterov wrote:
>>>>
>>>> I have to admit, I can't say I like this very much. OK, if we need
>>>> this, can't we just put something into, say, signal->flags so that
>>>> copy_process can check and create the new namespace.
>>>>
>>>> Also. I remember, I already saw something like this and google found
>>>> my questions. I didn't actually read the new version, perhaps my
>>>> concerns were already answered...
>>>>
>>>> But what if the task T does unshare(CLONE_NEWPID) and then, say,
>>>> pthread_create() ? Unless I missed something, the new thread won't
>>>> be able to see T ?
>>>
>>> Right. Is it really a problem ? I mean it is a weird use case where we
>>> fall in a weird situation.
>>
>> But this is really weird! How it is possible that the parent can't see
>> its own child? No matter which thread did fork(), the new process is

>
> Hmmm... I guess you mean the opposite. The way pid namespaces are
> nested, parents always see their children.

Well, yes. But it can't see this child using the same pid number,
unless I missed something.

> But indeed, the child thread
> can't see its group leader and that's kind of unusual.

This too. And to me this is more "kind of buggy". But yes, I am
biased because I dislike this approach in general ;)

And, once again, this patch also lacks the necessary s/nsproxy/atcive_pid_ns/
changes.

Anyway. It is very possible I missed something. As I said, I didn't
actually read this version and I forgot all I knew about this change
before.

But afaics this patch is buggy in its current form.

Oleg.

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

---

Subject: Re: [PATCH 1/1][3rd resend] sys_unshare: remove the dead CLONE_THREAD/SIGHAND/VM code
Posted by serge on Mon, 21 Feb 2011 00:17:43 GMT
View Forum Message <> Reply to Message

Quoting Oleg Nesterov (oleg@redhat.com):
> Cleanup: kill the dead code which does nothing but complicates the code
> and confuses the reader.
>
> sys_unshare(CLONE_THREAD/SIGHAND/VM) is not really implemented, and I doubt
> very much it will ever work. At least, nobody even tried since the original
> "unshare system call -v5: system call handler function" commit
> 99d1419d96d7df9cfa56bc977810be831bd5ef64 was applied more than 4 years ago.
>
> And the code is not consistent. unshare_thread() always fails unconditionally,
> while unshare_sighand() and unshare_vm() pretend to work if there is nothing
> to unshare.
>

> Remove unshare_thread(), unshare_sighand(), unshare_vm() helpers and related
> variables and add a simple CLONE_THREAD | CLONE_SIGHAND| CLONE_VM check into
> check_unshare_flags().
>
> Also, move the "CLONE_NEWNS needs CLONE_FS" check from check_unshare_flags()
> to sys_unshare(). This looks more consistent and matches the similar
> do_sysvsem check in sys_unshare().
>
> Note: with or without this patch "atomic_read(mm->mm_users) > 1" can give
> a false positive due to get_task_mm().
>
> Signed-off-by: Oleg Nesterov <oleg@redhat.com>
> Acked-by: Roland McGrath <roland@redhat.com>

Yes, please.

Acked-by: Serge Hallyn <serge.hallyn@canonical.com>

thanks,
-serge

> ---
>
> kernel/fork.c | 123 ++++++++++++--
> 1 file changed, 25 insertions(+), 98 deletions(-)
>
> --- 2.6.37/kernel/fork.c~unshare-killcrap 2010-11-05 18:03:28.000000000 +0100
> +++ 2.6.37/kernel/fork.c 2010-11-05 18:09:52.000000000 +0100
> @@ -1522,38 +1522,24 @@ void __init proc_caches_init(void)
> }
>
> /*
> - * Check constraints on flags passed to the unshare system call and
> - * force unsharing of additional process context as appropriate.
> + * Check constraints on flags passed to the unshare system call.
>   */
> -static void check_unshare_flags(unsigned long *flags_ptr)
> +static int check_unshare_flags(unsigned long unshare_flags)
> {
> + if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
> +   CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
> +   CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWNET))
> + return -EINVAL;
>   /*
> - * If unsharing a thread from a thread group, must also
> - * unshare vm.
> - */

> - if (*flags_ptr & CLONE_THREAD)
> - *flags_ptr |= CLONE_VM;
> -
> - /*
> - * If unsharing vm, must also unshare signal handlers.
> - */
> - if (*flags_ptr & CLONE_VM)
> - *flags_ptr |= CLONE_SIGHAND;
> -
> - /*
> - * If unsharing namespace, must also unshare filesystem information.
> + * Not implemented, but pretend it works if there is nothing to
> + * unshare. Note that unsharing CLONE_THREAD or CLONE_SIGHAND
> + * needs to unshare vm.
>   */
> - if (*flags_ptr & CLONE_NEWNS)
> - *flags_ptr |= CLONE_FS;
> -}
> -
> -/*
> - * Unsharing of tasks created with CLONE_THREAD is not supported yet
> - */
> -static int unshare_thread(unsigned long unshare_flags)
> -{
> - if (unshare_flags & CLONE_THREAD)
> - return -EINVAL;
> + if (unshare_flags & (CLONE_THREAD | CLONE_SIGHAND | CLONE_VM)) {
> + /* FIXME: get_task_mm() increments ->mm_users */
> + if (atomic_read(&current->mm->mm_users) > 1)
> +   return -EINVAL;
> + }
>
>   return 0;
> }
> @@ -1580,34 +1566,6 @@ static int unshare_fs(unsigned long unsh
> }
>
> /*
> - * Unsharing of sighand is not supported yet
> - */
> -static int unshare_sighand(unsigned long unshare_flags, struct sighand_struct **new_sighp)
> -{
> - struct sighand_struct *sigh = current->sighand;
> -
> - if ((unshare_flags & CLONE_SIGHAND) && atomic_read(&sigh->count) > 1)
> - return -EINVAL;
> - else
> - return 0;

```
> -}
> -
> -/*
> - * Unshare vm if it is being shared
> - */
> -static int unshare_vm(unsigned long unshare_flags, struct mm_struct **new_mmp)
> -{
> - struct mm_struct *mm = current->mm;
> -
> - if ((unshare_flags & CLONE_VM) &&
> -     (mm && atomic_read(&mm->mm_users) > 1)) {
> -  return -EINVAL;
> - }
> -
> - return 0;
> -}
> -
> -/*
>    * Unshare file descriptor table if it is being shared
>    */
>  static int unshare_fd(unsigned long unshare_flags, struct files_struct **new_fdp)
> @@ -1635,45 +1593,37 @@ static int unshare_fd(unsigned long unsh
>    */
>  SYSCALL_DEFINE1(unshare, unsigned long, unshare_flags)
>  {
> - int err = 0;
>    struct fs_struct *fs, *new_fs = NULL;
> - struct sighand_struct *new_sigh = NULL;
> - struct mm_struct *mm, *new_mm = NULL, *active_mm = NULL;
>    struct files_struct *fd, *new_fd = NULL;
>    struct nsproxy *new_nsproxy = NULL;
>    int do_sysvsem = 0;
> + int err;
>
> - check_unshare_flags(&unshare_flags);
> -
> - /* Return -EINVAL for all unsupported flags */
> - err = -EINVAL;
> - if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
> -     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
> -     CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWNET))
> + err = check_unshare_flags(unshare_flags);
> + if (err)
>     goto bad_unshare_out;
>
>    /*
> +  * If unsharing namespace, must also unshare filesystem information.
> +  */
```

```
> + if (unshare_flags & CLONE_NEWNS)
> +  unshare_flags |= CLONE_FS;
> + /*
>    * CLONE_NEWIPC must also detach from the undolist: after switching
>    * to a new ipc namespace, the semaphore arrays from the old
>    * namespace are unreachable.
>    */
>   if (unshare_flags & (CLONE_NEWIPC|CLONE_SYSVSEM))
>    do_sysvsem = 1;
> - if ((err = unshare_thread(unshare_flags)))
> -  goto bad_unshare_out;
>   if ((err = unshare_fs(unshare_flags, &new_fs)))
> -  goto bad_unshare_cleanup_thread;
> - if ((err = unshare_sighand(unshare_flags, &new_sigh)))
> -  goto bad_unshare_cleanup_fs;
> - if ((err = unshare_vm(unshare_flags, &new_mm)))
> -  goto bad_unshare_cleanup_sigh;
> +  goto bad_unshare_out;
>   if ((err = unshare_fd(unshare_flags, &new_fd)))
> -  goto bad_unshare_cleanup_vm;
> +  goto bad_unshare_cleanup_fs;
>   if ((err = unshare_nsproxy_namespaces(unshare_flags, &new_nsproxy,
>     new_fs)))
>    goto bad_unshare_cleanup_fd;
>
> - if (new_fs ||  new_mm || new_fd || do_sysvsem || new_nsproxy) {
> + if (new_fs || new_fd || do_sysvsem || new_nsproxy) {
>    if (do_sysvsem) {
>     /*
>      * CLONE_SYSVSEM is equivalent to sys_exit().
> @@ -1699,19 +1649,6 @@ SYSCALL_DEFINE1(unshare, unsigned long,
>     spin_unlock(&fs->lock);
>    }
>
> -  if (new_mm) {
> -   mm = current->mm;
> -   active_mm = current->active_mm;
> -   current->mm = new_mm;
> -   current->active_mm = new_mm;
> -   if (current->signal->oom_score_adj == OOM_SCORE_ADJ_MIN) {
> -    atomic_dec(&mm->oom_disable_count);
> -    atomic_inc(&new_mm->oom_disable_count);
> -   }
> -   activate_mm(active_mm, new_mm);
> -   new_mm = mm;
> -  }
> -
>    if (new_fd) {
```

>   fd = current->files;
>   current->files = new_fd;
> @@ -1728,20 +1665,10 @@ bad_unshare_cleanup_fd:
>   if (new_fd)
>   put_files_struct(new_fd);
>
> -bad_unshare_cleanup_vm:
> - if (new_mm)
> - mmput(new_mm);
> -
> -bad_unshare_cleanup_sigh:
> - if (new_sigh)
> - if (atomic_dec_and_test(&new_sigh->count))
> -   kmem_cache_free(sighand_cachep, new_sigh);
> -
> bad_unshare_cleanup_fs:
>   if (new_fs)
>   free_fs_struct(new_fs);
>
> -bad_unshare_cleanup_thread:
> bad_unshare_out:
>   return err;
> }
>
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containe rs

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

---

Subject: Re: [PATCH 2/2] pidns: Support unsharing the pid namespace.
Posted by Rob Landley on Thu, 24 Feb 2011 01:12:46 GMT
View Forum Message <> Reply to Message

On 02/15/2011 10:53 AM, Daniel Lezcano wrote:
> From: Eric W. Biederman <ebiederm@xmission.com>
>
> - Allow CLONEW_NEWPID into unshare. - Pass both nsproxy->pid_ns and
> task_active_pid_ns to copy_pid_ns As they can now be different.
>
> Unsharing of the pid namespace unlike unsharing of other namespaces
> does not take effect immediately. Instead it affects the children
> created with fork and clone. The first of these children becomes the
> init process of the new pid namespace, the rest become oddball

> children of pid 0. From the point of view of the new pid namespace
> the process that created it is pid 0, as it's pid does not map.
>
> A couple of different semantics were considered but this one was
> settled on because it is easy to implement and it is usable from pam
> modules. The core reasons for the existence of unshare.

Hmmm...

The userspace semantics I expected were for unshare(CLONE_NEWPID) to:

A) make the current process be PID 1 in the new namespace.

B) reparent_to_init() any existing children as if the process that
called unshare() had exited. (Because those children are not in the new
PID namespace.)

Is there a reason to implement it in some way other than that?

```
 if (!fork) {
   unshare(CLONE_NEWUSER);
   exec();
 }
```

> I took a survey of the callers of pam modules and the following
> appears to be a representative sample of their logic. { setup stuff
> include pam child = fork(); if (!child) { setuid() exec /bin/bash }
> waitpid(child);
>
> pam and other cleanup }

And calling unshare() right before calling setuid() seems the logical
thing to do there...?

Currently unshare() works like chroot(). You're making it act like
vfork() where the process that did this is in a strange halfway state
until it creates new children at which point magic happens. I don't
understand why this is an improvement, especially since none of the
other flags you can feed to unshare do that.

> As you can see there is a fork to create the unprivileged user space
> process. Which means that the unprivileged user space process will
> appear as pid 1 in the new pid namespace.

Meaning the process that called unshare() becomes the idle task? Or
this process isn't actually in the new PID namespace?

> Further most login processes do not cope with extraneous children

> which means shifting the duty of reaping extraneous child process to
> the creator of those extraneous children makes the system more
> comprehensible.

We already have reparent_to_init() happening when a process dies. That can't be adapted/reused?

> The practical reason for this set of pid namespace semantics is that
> it is simple to implement and verify they work correctly. Whereas an
> implementation that requires changing the struct pid on a process
> comes with a lot more races and pain. Not the least of which is that
> glibc caches getpid().

So unshare() in the libc needs to flush that cache. Presumably a one line patch.

> These semantics are implemented by having two notions of the pid
> namespace of a process. There is task_active_pid_ns which is the pid
> namspace the process was created with and the pid namespace that all
> pids are presented to that process in. The task_active_pid_ns is
> stored in the struct pid of the task.

Having two PID namespaces for each process is the simple answer?

> There is the pid namespace that will be used for children that pid
> namespace is stored in task->nsproxy->pid_ns.
>
> There is one really nasty corner case in all of this. Which pid
> namespace are you in if your parent unshared it's pid namespace and
> then on clone you also unshare the pid namespace. To me there are
> only two possible answers. Either the cases is so bizarre and we
> deny it completely. or the new pid namespace is a descendent of our
> parent's active pid namespace, and we ignore the
> task->nsproxy->pid_ns.
>
> To that end I have modified copy_pid_ns to take both of these pid
> namespaces. The active pid namespace and the default pid namespace
> of children. Allowing me to simply implement unsharing a pid
> namespace in clone after already unsharing a pid namespace with
> unshare.

If clone creates a new namespace, and unshare() discard that namespace and creates another new one, presumably the first one's reference count will go to zero if no processes are in it?

Also, when the PID 1 of a namespace leaves that namespace (generally by exiting), all the children get killed.

I thought the one nasty corner case is that the parent of PID 1 isn't
(ever) in the current PID namespace, so reference counting and list
membership gets a little funky. (I still need to read more about that...)

Rob

_____

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