Subject: [RFC][PATCH 0/15] Pid namespaces
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:45:15 GMT
View Forum Message <> Reply to Message

(Comment is taken from Sukadev's patchset-v3)

A pid namespace is a "view" of a particular set of tasks on the system.
They work in a similar way to filesystem namespaces. A file (or a process)
can be accessed in multiple namespaces, but it may have a different name
in each. In a filesystem, this name might be /etc/passwd in one namespace,
but /chroot/etc/passwd in another.

For processes, a process may have pid 1234 in one namespace, but be pid 1
in another. This allows new pid namespaces to have basically arbitrary
pids, and not have to worry about what pids exist in other namespaces.
This is essential for checkpoint/restart where a restarted process's pid
might collide with an existing process on the system's pid.

In this particular implementation, pid namespaces have a parent-child
relationship, just like processes. A process in a pid namespace may see
all of the processes in the same namespace, as well as all of the processes
in all of the namespaces which are children of its namespace. Processes may
not, however, see others which are in their parent's namespace, but not in
their own. The same goes for sibling namespaces.

This set is based on my patches, I sent before, but it includes some comments
and patches that I received from Sukadev. Sukadev, please, add your Acked-by,
Signed-off-by or From, to patches you want (everybody is also welcome :)).

The set is based on 2.6.23-rc1-mm1, which already has some preparation patches
for pid namespaces. After the review and fixing all the comments, this set
will be benchmarked and sent to Andrew for inclusion in -mm tree.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

Subject: [PATCH 1/15] Move exit_task_namespaces()
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:46:28 GMT
View Forum Message <> Reply to Message

Make task release its namespaces after it has reparented all his
children to child_reaper, but before it notifies its parent about
its death.

The reason to release namespaces after reparenting is that when task
exits it may send a signal to its parent (SIGCHLD), but if the parent
has already exited its namespaces there will be no way to decide what

pid to dever to him - parent can be from different namespace.

The reason to release namespace before notifying the parent it that
when task sends a SIGCHLD to parent it can call wait() on this taks
and release it. But releasing the mnt namespace implies dropping
of all the mounts in the mnt namespace and NFS expects the task to
have valid sighand pointer.

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

---

 exit.c | 5 ++++-
 1 files changed, 4 insertions(+), 1 deletion(-)

diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c linux-2.6.23-rc1-mm1-7/kernel/exit.c
--- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26 16:36:37.000000000 +0400
@@ -788,6 +804,10 @@ static void exit_notify(struct task_stru
  BUG_ON(!list_empty(&tsk->children));
  BUG_ON(!list_empty(&tsk->ptrace_children));

+ write_unlock_irq(&tasklist_lock);
+ exit_task_namespaces(tsk);
+ write_lock_irq(&tasklist_lock);
+
  /*
   * Check to see if any process groups have become orphaned
   * as a result of our exiting, and if they have any stopped
@@ -999,7 +1021,6 @@ fastcall NORET_TYPE void do_exit(long co

  tsk->exit_code = code;
  proc_exit_connector(tsk);
- exit_task_namespaces(tsk);
  exit_notify(tsk);
 #ifdef CONFIG_NUMA
  mpol_free(tsk->mempolicy);

Subject: [PATCH 2/15] Introduce MS_KERNMOUNT flag
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:47:23 GMT
View Forum Message <> Reply to Message

This flag tells the .get_sb callback that this is a kern_mount() call
so that it can trust *data pointer to be valid in-kernel one.

Running a few steps forward - this will be needed for proc to create the
superblock and store a valid pid namespace on it during the namespace

creation. The reason, why the namespace cannot live without proc mount
is described in the appropriate patch.

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

```
 fs/namespace.c     |    3 ++-
 fs/super.c         |    6 +++---
 include/linux/fs.h |    4 +++-
 3 files changed, 8 insertions(+), 5 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/fs/namespace.c linux-2.6.23-rc1-mm1-7/fs/namespace.c
--- linux-2.6.23-rc1-mm1.orig/fs/namespace.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/namespace.c 2007-07-26 16:36:36.000000000 +0400
@@ -1579,7 +1579,8 @@ long do_mount(char *dev_name, char *dir_
   mnt_flags |= MNT_NOMNT;

  flags &= ~(MS_NOSUID | MS_NOEXEC | MS_NODEV | MS_ACTIVE |
-     MS_NOATIME | MS_NODIRATIME | MS_RELATIME | MS_NOMNT);
+     MS_NOATIME | MS_NODIRATIME | MS_RELATIME |
+     MS_NOMNT | MS_KERNMOUNT);

  /* ... and get the mountpoint */
  retval = path_lookup(dir_name, LOOKUP_FOLLOW, &nd);
diff -upr linux-2.6.23-rc1-mm1.orig/fs/super.c linux-2.6.23-rc1-mm1-7/fs/super.c
--- linux-2.6.23-rc1-mm1.orig/fs/super.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/super.c 2007-07-26 16:36:36.000000000 +0400
@@ -944,9 +944,9 @@ do_kern_mount(const char *fstype, int fl
  return mnt;
 }

-struct vfsmount *kern_mount(struct file_system_type *type)
+struct vfsmount *kern_mount_data(struct file_system_type *type, void *data)
 {
- return vfs_kern_mount(type, 0, type->name, NULL);
+ return vfs_kern_mount(type, MS_KERNMOUNT, type->name, data);
 }

-EXPORT_SYMBOL(kern_mount);
+EXPORT_SYMBOL_GPL(kern_mount_data);
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/fs.h linux-2.6.23-rc1-mm1-7/include/linux/fs.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/fs.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/fs.h 2007-07-26 16:36:36.000000000 +0400
@@ -129,6 +129,7 @@ extern int dir_notify_enable;
 #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
 #define MS_SETUSER (1<<23) /* set mnt_uid to current user */
 #define MS_NOMNT (1<<24) /* don't allow unprivileged submounts */
```

```
+#define MS_KERNMOUNT (1<<25) /* this is a kern_mount call */
 #define MS_ACTIVE (1<<30)
 #define MS_NOUSER (1<<31)

@@ -1459,7 +1460,8 @@ void unnamed_dev_init(void);

 extern int register_filesystem(struct file_system_type *);
 extern int unregister_filesystem(struct file_system_type *);
-extern struct vfsmount *kern_mount(struct file_system_type *);
+extern struct vfsmount *kern_mount_data(struct file_system_type *, void *data);
+#define kern_mount(type) kern_mount_data(type, NULL)
 extern int may_umount_tree(struct vfsmount *);
 extern int may_umount(struct vfsmount *);
 extern void umount_tree(struct vfsmount *, int, struct list_head *);
```

Subject: [PATCH 3/15] kern_siginfo helper
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:48:15 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

get_signal_to_deliver() checks to ensure that /sbin/init does not
receive any unwanted signals. With implementation of multiple pid
namespaces, we need to extend this check for all "container-init"
processes (processes with pid == 1 in the pid namespace)

IOW, A container-init process, must not receive unwanted signals from
within the container. However, the container-init must receive and
honor any signals it receives from an ancestor pid namespace (i.e it
must appear like a normal process in its parent namespace).

i.e for correct processing of the signal, either:

- the recepient of the signal (in get_signal_to_deliver()) must
  have some information (such as pid namespace) of the sender

- or the sender of the signal (in send_signal()) should know
  whether the signal is an "unwanted" signal from the recepient's
  POV.

This patch provides a simple mechanism, to pass additional information from
the sender to the recepient (from send_signal() to get_signal_to_deliver()).

This patch is just a helper and a follow-on patch will use this infrastructure
to actually pass in information about the sender.

Implementation note:

Most signal interfaces in the kernel operate on 'siginfo_t' data
structure which is user-visible and cannot be easily modified/
extended. So this patch defines a wrapper, 'struct kern_siginfo',
around the 'siginfo_t' and allows extending this wrapper for future
needs.

TODO: This is more an exploratory patch and modifies only interfaces
necessary to implement correct signal semantics in pid namespaces.

If the approach is feasible, we could consistently use 'kern_siginfo'
in other signal interfaces and possibly in 'struct sigqueue'.

We could modify dequeue_signal() to directly work with 'kern_siginfo'
and remove dequeue_signal_kern_info().

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

---

include/linux/signal.h | 7 +++++++
kernel/signal.c | 38 ++++++++++++++++++++++++++++++++------
2 files changed, 38 insertions(+), 7 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/signal.h
linux-2.6.23-rc1-mm1-7/include/linux/signal.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/signal.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/signal.h 2007-07-26 16:36:37.000000000 +0400
@@ -7,6 +7,13 @@
#ifdef __KERNEL__
#include <linux/list.h>

+struct kern_siginfo {
+ siginfo_t *info;
+ int flags;
+};
+
+#define KERN_SIGINFO_CINIT 0x1 /* True iff signalling container-init */
+
/*
 * Real Time signals may be queued.
 */
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/signal.c linux-2.6.23-rc1-mm1-7/kernel/signal.c
--- linux-2.6.23-rc1-mm1.orig/kernel/signal.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/signal.c 2007-07-26 16:36:37.000000000 +0400
@@ -299,10 +299,12 @@ unblock_all_signals(void)
 spin_unlock_irqrestore(&current->sighand->siglock, flags);
}

```
-static int collect_signal(int sig, struct sigpending *list, siginfo_t *info)
+static int collect_signal(int sig, struct sigpending *list,
+  struct kern_siginfo *kinfo)
 {
  struct sigqueue *q, *first = NULL;
  int still_pending = 0;
+ siginfo_t *info = kinfo->info;

  if (unlikely(!sigismember(&list->signal, sig)))
   return 0;
@@ -343,7 +348,7 @@ static int collect_signal(int sig, struc
 }

 static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
-   siginfo_t *info)
+   struct kern_siginfo *kinfo)
 {
  int sig = next_signal(pending, mask);

@@ -357,7 +364,7 @@ static int __dequeue_signal(struct sigpe
    }
   }

-  if (!collect_signal(sig, pending, info))
+  if (!collect_signal(sig, pending, kinfo))
    sig = 0;
  }

@@ -370,18 +377,20 @@ static int __dequeue_signal(struct sigpe
  *
  * All callers have to hold the siglock.
  */
-int dequeue_signal(struct task_struct *tsk, sigset_t *mask, siginfo_t *info)
+int dequeue_signal_kern_info(struct task_struct *tsk, sigset_t *mask,
+  struct kern_siginfo *kinfo)
 {
  int signr = 0;
+ siginfo_t *info = kinfo->info;

  /* We only dequeue private signals from ourselves, we don't let
   * signalfd steal them
   */
  if (tsk == current)
-  signr = __dequeue_signal(&tsk->pending, mask, info);
+  signr = __dequeue_signal(&tsk->pending, mask, kinfo);
  if (!signr) {
   signr = __dequeue_signal(&tsk->signal->shared_pending,
```

```
-      mask, info);
+      mask, kinfo);
   /*
    * itimer signal ?
    *
@@ -441,6 +450,22 @@ int dequeue_signal(struct task_struct *t
 }

 /*
+ * Dequeue a signal and return the element to the caller, which is
+ * expected to free it.
+ *
+ * All callers have to hold the siglock.
+ */
+int dequeue_signal(struct task_struct *tsk, sigset_t *mask, siginfo_t *info)
+{
+ struct kern_siginfo kinfo;
+
+ kinfo.info = info;
+ kinfo.flags = 0;
+
+ return dequeue_signal_kern_info(tsk, mask, &kinfo);
+}
+
+/*
  * Tell a process that it has a new active signal..
  *
  * NOTE! we rely on the previous spin_lock to
@@ -1779,6 +1873,10 @@ int get_signal_to_deliver(siginfo_t *inf
 {
  sigset_t *mask = &current->blocked;
  int signr = 0;
+ struct kern_siginfo kinfo;
+
+ kinfo.info = info;
+ kinfo.flags = 0;

  try_to_freeze();

@@ -1791,7 +1889,7 @@ relock:
       handle_group_stop())
    goto relock;

-  signr = dequeue_signal(current, mask, info);
+  signr = dequeue_signal_kern_info(current, mask, &kinfo);

   if (!signr)
    break; /* will return 0 */
```

Subject: [PATCH 4/15] Make proc_flust_task() flush entries from multiple proc trees
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:48:59 GMT
View Forum Message <> Reply to Message

Since a task will appear in more than one proc tree we need to shrink many
trees. For this case we pass the struct pid to proc_flush_task() and shrink
the mounts of all the namespaces this pid belongs to.

The NULL passed to it means that only global mount is to be flushed.
This is a preparation for a real flushing patch.

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

---

 fs/proc/base.c | 18 ++++++++++++++---
 include/linux/proc_fs.h | 6 ++++--
 kernel/exit.c | 2 +-
 3 files changed, 20 insertions(+), 6 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/fs/proc/base.c linux-2.6.23-rc1-mm1-7/fs/proc/base.c
--- linux-2.6.23-rc1-mm1.orig/fs/proc/base.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/proc/base.c 2007-07-26 16:36:37.000000000 +0400
@@ -74,6 +74,7 @@
 #include <linux/nsproxy.h>
 #include <linux/oom.h>
 #include <linux/elf.h>
+#include <linux/pid_namespace.h>
 #include "internal.h"

 /* NOTE:
@@ -2115,7 +2116,7 @@ static const struct inode_operations pro
  *   that no dcache entries will exist at process exit time it
  *   just makes it very unlikely that any will persist.
  */
-void proc_flush_task(struct task_struct *task)
+static void proc_flush_task_mnt(struct task_struct *task, struct vfsmount *mnt)
 {
  struct dentry *dentry, *leader, *dir;
  char buf[PROC_NUMBUF];
@@ -2123,7 +2124,7 @@ void proc_flush_task(struct task_struct

  name.name = buf;
  name.len = snprintf(buf, sizeof(buf), "%d", task->pid);
- dentry = d_hash_and_lookup(proc_mnt->mnt_root, &name);
+ dentry = d_hash_and_lookup(mnt->mnt_root, &name);
  if (dentry) {
   shrink_dcache_parent(dentry);
   d_drop(dentry);

```
@@ -2135,7 +2136,7 @@ void proc_flush_task(struct task_struct

  name.name = buf;
  name.len = snprintf(buf, sizeof(buf), "%d", task->tgid);
- leader = d_hash_and_lookup(proc_mnt->mnt_root, &name);
+ leader = d_hash_and_lookup(mnt->mnt_root, &name);
  if (!leader)
   goto out;

@@ -2161,6 +2162,17 @@ out:
  return;
 }

+/*
+ * when flushing dentries from proc one need to flush them from global
+ * proc (proc_mnt) and from all the namespaces' procs this task was seen
+ * in. this call is supposed to make all this job.
+ */
+
+void proc_flush_task(struct task_struct *task, struct pid *pid)
+{
+ proc_flush_task_mnt(task, proc_mnt);
+}
+
 static struct dentry *proc_pid_instantiate(struct inode *dir,
        struct dentry * dentry,
        struct task_struct *task, const void *ptr)
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/proc_fs.h
linux-2.6.23-rc1-mm1-7/include/linux/proc_fs.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/proc_fs.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/proc_fs.h 2007-07-26 16:36:36.000000000 +0400
@@ -111,7 +111,7 @@ extern void proc_misc_init(void);

 struct mm_struct;

-void proc_flush_task(struct task_struct *task);
+void proc_flush_task(struct task_struct *task, struct pid *pid);
 struct dentry *proc_pid_lookup(struct inode *dir, struct dentry * dentry, struct nameidata *);
 int proc_pid_readdir(struct file * filp, void * dirent, filldir_t filldir);
 unsigned long task_vsize(struct mm_struct *);
@@ -223,7 +227,9 @@ static inline void proc_net_remove(const
 #define proc_net_create(name, mode, info) ({ (void)(mode), NULL; })
 static inline void proc_net_remove(const char *name) {}

-static inline void proc_flush_task(struct task_struct *task) { }
+static inline void proc_flush_task(struct task_struct *task, struct pid *pid)
+{
+}
```

```
 static inline struct proc_dir_entry *create_proc_entry(const char *name,
  mode_t mode, struct proc_dir_entry *parent) { return NULL; }
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c linux-2.6.23-rc1-mm1-7/kernel/exit.c
--- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26 16:36:37.000000000 +0400
@@ -184,7 +199,7 @@ repeat:
  }

  write_unlock_irq(&tasklist_lock);
- proc_flush_task(p);
+ proc_flush_task(p, NULL);
  release_thread(p);
  call_rcu(&p->rcu, delayed_put_task_struct);
```

Subject: [PATCH 5/15] Introduce struct upid
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:49:42 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

Since task will be visible from different pid namespaces each of them
have to be addressed by multiple pids. struct upid is to store the
information about which id refers to which namespace.

The constuciton looks like this. Each struct pid carried the reference
counter and the list of tasks attached to this pid. At its end it has
a variable length array of struct upid-s. Each struct upid has a numerical
id (pid itself), pointer to the namespace, this ID is valid in and is
hashed into a pid_hash for searching the pids.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
 init_task.h |    9 ++++++---
 pid.h       |   12 ++++++++++---
 2 files changed, 15 insertions(+), 6 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/init_task.h
linux-2.6.23-rc1-mm1-7/include/linux/init_task.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/init_task.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/init_task.h 2007-07-26 16:36:36.000000000 +0400
@@ -93,15 +93,18 @@ extern struct group_info init_groups;

 #define INIT_STRUCT_PID {      \
```

```
  .count   = ATOMIC_INIT(1),    \
- .nr  = 0,       \
- /* Don't put this struct pid in pid_hash */   \
- .pid_chain = { .next = NULL, .pprev = NULL },  \
  .tasks  = {      \
   { .first = &init_task.pids[PIDTYPE_PID].node },  \
   { .first = &init_task.pids[PIDTYPE_PGID].node }, \
   { .first = &init_task.pids[PIDTYPE_SID].node },  \
  },        \
  .rcu  = RCU_HEAD_INIT,    \
+ .level  = 0,      \
+ .numbers = { {      \
+  .nr  = 0,     \
+  .ns  = &init_pid_ns,    \
+  .pid_chain = { .next = NULL, .pprev = NULL }, \
+ }, }       \
 }

 #define INIT_PID_LINK(type)      \
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid.h linux-2.6.23-rc1-mm1-7/include/linux/pid.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
@@ -40,15 +40,21 @@ enum pid_type
  * processes.
  */

-struct pid
-{
- atomic_t count;
+struct upid {
  /* Try to keep pid_chain in the same cacheline as nr for find_pid */
  int nr;
+ struct pid_namespace *ns;
  struct hlist_node pid_chain;
+};
+
+struct pid
+{
+ atomic_t count;
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
  struct rcu_head rcu;
+ int level;
+ struct upid numbers[1];
 };

 extern struct pid init_struct_pid;
```

Subject: [PATCH 6/15] Make alloc_pid(), free_pid() and put_pid() work with struct upid
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:50:22 GMT
View Forum Message <> Reply to Message

Each struct upid element of struct pid has to be initialized properly,
i.e. its nr mst be allocated from appropriate pidmap and it must be
inserted into the hash.

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

```
 include/linux/pid.h |    2 +-
 kernel/pid.c        |   52 ++++++++++++++++++++++++++++++++++++++---------------
 2 files changed, 38 insertions(+), 16 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid.h linux-2.6.23-rc1-mm1-7/include/linux/pid.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
@@ -83,7 +92,7 @@ extern void FASTCALL(detach_pid(struct t
 extern struct pid *find_get_pid(int nr);
 extern struct pid *find_ge_pid(int nr);

-extern struct pid *alloc_pid(void);
+extern struct pid *alloc_pid(struct pid_namespace *ns);
 extern void FASTCALL(free_pid(struct pid *pid));

 static inline pid_t pid_nr(struct pid *pid)
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -28,7 +28,8 @@
 #include <linux/pid_namespace.h>
 #include <linux/init_task.h>

-#define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
+#define pid_hashfn(nr, ns) \
+ hash_long((unsigned long)nr + (unsigned long)ns, pidhash_shift)
 static struct hlist_head *pid_hash;
 static int pidhash_shift;
 struct pid init_struct_pid = INIT_STRUCT_PID;
@@ -187,11 +202,13 @@ fastcall void put_pid(struct pid *pid)
  if (!pid)
   return;

- /* FIXME - this must be the namespace this pid lives in */
- ns = &init_pid_ns;
+ ns = pid->numbers[pid->level].ns;
```

```
  if ((atomic_read(&pid->count) == 1) ||
-      atomic_dec_and_test(&pid->count))
+      atomic_dec_and_test(&pid->count)) {
   kmem_cache_free(ns->pid_cachep, pid);
+  if (ns != &init_pid_ns)
+   put_pid_ns(ns);
+ }
 }
 EXPORT_SYMBOL_GPL(put_pid);

@@ -204,45 +221,64 @@ static void delayed_put_pid(struct rcu_h
 fastcall void free_pid(struct pid *pid)
 {
  /* We can be called with write_lock_irq(&tasklist_lock) held */
+ int i;
  unsigned long flags;

  spin_lock_irqsave(&pidmap_lock, flags);
- hlist_del_rcu(&pid->pid_chain);
+ for (i = 0; i <= pid->level; i++)
+  hlist_del_rcu(&pid->numbers[i].pid_chain);
  spin_unlock_irqrestore(&pidmap_lock, flags);

- free_pidmap(&init_pid_ns, pid->nr);
+ for (i = 0; i <= pid->level; i++)
+  free_pidmap(pid->numbers[i].ns, pid->numbers[i].nr);
+
  call_rcu(&pid->rcu, delayed_put_pid);
 }

-struct pid *alloc_pid(void)
+struct pid *alloc_pid(struct pid_namespace *ns)
 {
  struct pid *pid;
  enum pid_type type;
- int nr = -1;
- struct pid_namespace *ns;
+ int i, nr;
+ struct pid_namespace *tmp;

- ns = task_active_pid_ns(current);
  pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
  if (!pid)
   goto out;

- nr = alloc_pidmap(ns);
- if (nr < 0)
-  goto out_free;
```

```
+ tmp = ns;
+ for (i = ns->level; i >= 0; i--) {
+  nr = alloc_pidmap(tmp);
+  if (nr < 0)
+   goto out_free;
+
+  pid->numbers[i].nr = nr;
+  pid->numbers[i].ns = tmp;
+  tmp = tmp->parent;
+ }

+ if (ns != &init_pid_ns)
+  get_pid_ns(ns);
+
+ pid->level = ns->level;
  atomic_set(&pid->count, 1);
- pid->nr = nr;
  for (type = 0; type < PIDTYPE_MAX; ++type)
   INIT_HLIST_HEAD(&pid->tasks[type]);

  spin_lock_irq(&pidmap_lock);
- hlist_add_head_rcu(&pid->pid_chain, &pid_hash[pid_hashfn(pid->nr)]);
+ for (i = pid->level; i >= 0; i--)
+  hlist_add_head_rcu(&pid->numbers[i].pid_chain,
+    &pid_hash[pid_hashfn(pid->numbers[i].nr,
+     pid->numbers[i].ns)]);
  spin_unlock_irq(&pidmap_lock);

 out:
  return pid;

 out_free:
+ for (i++; i <= ns->level; i++)
+  free_pidmap(pid->numbers[i].ns, pid->numbers[i].nr);
+
  kmem_cache_free(ns->pid_cachep, pid);
  pid = NULL;
  goto out;
```

Subject: [PATCH 7/15] Helpers to obtain pid numbers
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:51:02 GMT
View Forum Message <> Reply to Message

When showing pid to user or getting the pid numerical id for in-kernel
use the value of this id may differ depending on the namespace.

This set of helpers is used to get the global pid nr, the virtual (i.e.

seen by task in its namespace) nr and the nr as it is seen from the
specified namespace.

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

```
 include/linux/pid.h   |   24 ++++++++++-
 include/linux/sched.h |  108 ++++++++++++++++++++++++++++++++++++++++++++++-----
 kernel/pid.c          |    8 +++
 3 files changed, 129 insertions(+), 11 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid.h linux-2.6.23-rc1-mm1-7/include/linux/pid.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
@@ -83,12 +92,34 @@ extern void FASTCALL(detach_pid(struct t

 extern struct pid *alloc_pid(struct pid_namespace *ns);
 extern void FASTCALL(free_pid(struct pid *pid));
+
+/*
+ * the helpers to get the pid's id seen from different namespaces
+ *
+ * pid_nr()    : global id, i.e. the id seen from the init namespace;
+ * pid_vnr()   : virtual id, i.e. the id seen from the namespace this pid
+ *               belongs to. this only makes sence when called in the
+ *               context of the task that belongs to the same namespace;
+ * pid_nr_ns() : id seen from the ns specified.
+ *
+ * see also task_xid_nr() etc in include/linux/sched.h
+ */

 static inline pid_t pid_nr(struct pid *pid)
 {
  pid_t nr = 0;
  if (pid)
-  nr = pid->nr;
+  nr = pid->numbers[0].nr;
+ return nr;
+}
+
+pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns);
+
+static inline pid_t pid_vnr(struct pid *pid)
+{
+ pid_t nr = 0;
+ if (pid)
+  nr = pid->numbers[pid->level].nr;
```

```
  return nr;
 }

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/sched.h
linux-2.6.23-rc1-mm1-7/include/linux/sched.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/sched.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/sched.h 2007-07-26 16:36:37.000000000 +0400
@@ -1221,16 +1222,6 @@ static inline int rt_task(struct task_st
  return rt_prio(p->prio);
 }

-static inline pid_t task_pgrp_nr(struct task_struct *tsk)
-{
- return tsk->signal->pgrp;
-}
-
-static inline pid_t task_session_nr(struct task_struct *tsk)
-{
- return tsk->signal->__session;
-}
-
 static inline void set_task_session(struct task_struct *tsk, pid_t session)
 {
  tsk->signal->__session = session;
@@ -1256,6 +1247,104 @@ static inline struct pid *task_session(s
  return task->group_leader->pids[PIDTYPE_SID].pid;
 }

+struct pid_namespace;
+
+/*
+ * the helpers to get the task's different pids as they are seen
+ * from various namespaces
+ *
+ * task_xid_nr()     : global id, i.e. the id seen from the init namespace;
+ * task_xid_vnr()    : virtual id, i.e. the id seen from the namespace the task
+ *                     belongs to. this only makes sence when called in the
+ *                     context of the task that belongs to the same namespace;
+ * task_xid_nr_ns()  : id seen from the ns specified;
+ *
+ * set_task_vxid()   : assigns a virtual id to a task;
+ *
+ * task_ppid_nr_ns() : the parent's id as seen from the namespace specified.
+ *                     the result depends on the namespace and whether the
+ *                     task in question is the namespace's init. e.g. for the
+ *                     namespace's init this will return 0 when called from
+ *                     the namespace of this init, or appropriate id otherwise.
+ *
```

```
+ *
+ * see also pid_nr() etc in include/linux/pid.h
+ */
+
+static inline pid_t task_pid_nr(struct task_struct *tsk)
+{
+ return tsk->pid;
+}
+
+static inline pid_t task_pid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return pid_nr_ns(task_pid(tsk), ns);
+}
+
+static inline pid_t task_pid_vnr(struct task_struct *tsk)
+{
+ return pid_vnr(task_pid(tsk));
+}
+
+
+static inline pid_t task_tgid_nr(struct task_struct *tsk)
+{
+ return tsk->tgid;
+}
+
+static inline pid_t task_tgid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return pid_nr_ns(task_tgid(tsk), ns);
+}
+
+static inline pid_t task_tgid_vnr(struct task_struct *tsk)
+{
+ return pid_vnr(task_tgid(tsk));
+}
+
+
+static inline pid_t task_pgrp_nr(struct task_struct *tsk)
+{
+ return tsk->signal->pgrp;
+}
+
+static inline pid_t task_pgrp_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return pid_nr_ns(task_pgrp(tsk), ns);
+}
```

```
+
+static inline pid_t task_pgrp_vnr(struct task_struct *tsk)
+{
+ return pid_vnr(task_pgrp(tsk));
+}
+
+
+static inline pid_t task_session_nr(struct task_struct *tsk)
+{
+ return tsk->signal->__session;
+}
+
+static inline pid_t task_session_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return pid_nr_ns(task_session(tsk), ns);
+}
+
+static inline pid_t task_session_vnr(struct task_struct *tsk)
+{
+ return pid_vnr(task_session(tsk));
+}
+
+
+static inline pid_t task_ppid_nr_ns(struct task_struct *tsk,
+  struct pid_namespace *ns)
+{
+ return pid_nr_ns(task_pid(rcu_dereference(tsk->real_parent)), ns);
+}
+
 /**
  * pid_alive - check that a task structure is not stale
  * @p: Task structure to be checked.
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -352,6 +436,14 @@ struct pid *find_get_pid(pid_t nr)
  return pid;
 }

+pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
+{
+ pid_t nr = 0;
+ if (pid && ns->level <= pid->level)
+  nr = pid->numbers[ns->level].nr;
+ return nr;
+}
+
```

```
 /*
  * Used by proc to find the first pid that is greater then or equal to nr.
  *
```

Subject: [PATCH 8/15] Helpers to find the task by its numerical ids
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:51:46 GMT
View Forum Message <> Reply to Message

When searching the task by numerical id on may need to find
it using global pid (as it is done now in kernel) or by its
virtual id, e.g. when sending a signal to a task from one
namespace the sender will specify the task's virtual id.

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

```
 include/linux/pid.h   |   16 +++++++++++++--
 include/linux/sched.h |   31 ++++++++++++++++++++++++++++--
 kernel/pid.c          |   32 ++++++++++++++++++++++----------
 3 files changed, 60 insertions(+), 19 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid.h linux-2.6.23-rc1-mm1-7/include/linux/pid.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
@@ -83,17 +92,29 @@ extern void FASTCALL(detach_pid(struct t
 extern void FASTCALL(transfer_pid(struct task_struct *old,
       struct task_struct *new, enum pid_type));

+struct pid_namespace;
+extern struct pid_namespace init_pid_ns;
+
 /*
  * look up a PID in the hash table. Must be called with the tasklist_lock
  * or rcu_read_lock() held.
+ *
+ * find_pid_ns() finds the pid in the namespace specified
+ * find_pid() find the pid by its global id, i.e. in the init namespace
+ * find_vpid() finr the pid by its virtual id, i.e. in the current namespace
+ *
+ * see also find_task_by_pid() set in include/linux/sched.h
  */
-extern struct pid *FASTCALL(find_pid(int nr));
+extern struct pid *FASTCALL(find_pid_ns(int nr, struct pid_namespace *ns));
+
+#define find_vpid(pid) find_pid_ns(pid, current->nsproxy->pid_ns)
+#define find_pid(pid) find_pid_ns(pid, &init_pid_ns)
```

```
 /*
  * Lookup a PID in the hash table, and return with it's count elevated.
  */
 extern struct pid *find_get_pid(int nr);
-extern struct pid *find_ge_pid(int nr);
+extern struct pid *find_ge_pid(int nr, struct pid_namespace *);

 extern struct pid *alloc_pid(struct pid_namespace *ns);
 extern void FASTCALL(free_pid(struct pid *pid));
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/sched.h
linux-2.6.23-rc1-mm1-7/include/linux/sched.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/sched.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/sched.h 2007-07-26 16:36:37.000000000 +0400
@@ -1453,8 +1542,35 @@ extern struct task_struct init_task;

 extern struct   mm_struct init_mm;

-#define find_task_by_pid(nr) find_task_by_pid_type(PIDTYPE_PID, nr)
-extern struct task_struct *find_task_by_pid_type(int type, int pid);
+extern struct pid_namespace init_pid_ns;
+
+/*
+ * find a task by one of its numerical ids
+ *
+ * find_task_by_pid_type_ns():
+ *      it is the most generic call - it finds a task by all id,
+ *      type and namespace specified
+ * find_task_by_pid_ns():
+ *      finds a task by its pid in the specified namespace
+ * find_task_by_pid_type():
+ *      finds a task by its global id with the specified type, e.g.
+ *      by global session id
+ * find_task_by_pid():
+ *      finds a task by its global pid
+ *
+ * see also find_pid() etc in include/linux/pid.h
+ */
+
+extern struct task_struct *find_task_by_pid_type_ns(int type, int pid,
+  struct pid_namespace *ns);
+
+#define find_task_by_pid_ns(nr, ns) \
+  find_task_by_pid_type_ns(PIDTYPE_PID, nr, ns)
+#define find_task_by_pid_type(type, nr) \
+  find_task_by_pid_type_ns(type, nr, &init_pid_ns)
+#define find_task_by_pid(nr)  \
+  find_task_by_pid_type(PIDTYPE_PID, nr)
```

```
+
 extern void __set_special_pids(pid_t session, pid_t pgrp);

 /* per-UID process charging. */
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -204,19 +221,20 @@ static void delayed_put_pid(struct rcu_h
  goto out;
 }

-struct pid * fastcall find_pid(int nr)
+struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
 {
  struct hlist_node *elem;
- struct pid *pid;
+ struct upid *pnr;
+
+ hlist_for_each_entry_rcu(pnr, elem,
+   &pid_hash[pid_hashfn(nr, ns)], pid_chain)
+  if (pnr->nr == nr && pnr->ns == ns)
+   return container_of(pnr, struct pid,
+     numbers[ns->level]);

- hlist_for_each_entry_rcu(pid, elem,
-   &pid_hash[pid_hashfn(nr)], pid_chain) {
-  if (pid->nr == nr)
-   return pid;
- }
  return NULL;
 }
-EXPORT_SYMBOL_GPL(find_pid);
+EXPORT_SYMBOL_GPL(find_pid_ns);

 /*
  * attach_pid() must be called with the tasklist_lock write-held.
@@ -318,12 +355,13 @@ struct task_struct * fastcall pid_task(s
 /*
  * Must be called under rcu_read_lock() or with tasklist_lock read-held.
  */
-struct task_struct *find_task_by_pid_type(int type, int nr)
+struct task_struct *find_task_by_pid_type_ns(int type, int nr,
+  struct pid_namespace *ns)
 {
- return pid_task(find_pid(nr), type);
+ return pid_task(find_pid_ns(nr, ns), type);
 }
```

```
-EXPORT_SYMBOL(find_task_by_pid_type);
+EXPORT_SYMBOL(find_task_by_pid_type_ns);

 struct pid *get_task_pid(struct task_struct *task, enum pid_type type)
 {
@@ -342,7 +426,7 @@ struct pid *find_get_pid(pid_t nr)
  struct pid *pid;

  rcu_read_lock();
- pid = get_pid(find_pid(nr));
+ pid = get_pid(find_vpid(nr));
  rcu_read_unlock();

  return pid;
@@ -352,15 +436,15 @@ struct pid *find_get_pid(pid_t nr)
  *
  * If there is a pid at nr this function is exactly the same as find_pid.
  */
-struct pid *find_ge_pid(int nr)
+struct pid *find_ge_pid(int nr, struct pid_namespace *ns)
 {
  struct pid *pid;

  do {
-  pid = find_pid(nr);
+  pid = find_pid_ns(nr, ns);
   if (pid)
    break;
-  nr = next_pidmap(task_active_pid_ns(current), nr);
+  nr = next_pidmap(ns, nr);
  } while (nr > 0);

  return pid;
```

Subject: [PATCH 9/15] Move alloc_pid() after the namespace is cloned
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:52:33 GMT
View Forum Message <> Reply to Message

When we create new namespace we will need to allocate the struct pid,
that will have one extra struct upid in array, comparing to the parent.
Thus we need to know the new namespace (if any) in alloc_pid() to init
this struct upid properly, so move the alloc_pid() call lower in
copy_process().

This is a fix for Sukadev's patch that moved the alloc_pid() call from
do_fork() into copy_process().

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

---

 fork.c | 59 ++---------------
 1 files changed, 38 insertions(+), 21 deletions(-)

```
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/fork.c linux-2.6.23-rc1-mm1-7/kernel/fork.c
--- linux-2.6.23-rc1-mm1.orig/kernel/fork.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/fork.c 2007-07-26 16:36:37.000000000 +0400
@@ -1028,16 +1029,9 @@ static struct task_struct *copy_process(
  if (p->binfmt && !try_module_get(p->binfmt->module))
   goto bad_fork_cleanup_put_domain;

- if (pid != &init_struct_pid) {
-  pid = alloc_pid();
-  if (!pid)
-   goto bad_fork_put_binfmt_module;
- }
-
  p->did_exec = 0;
  delayacct_tsk_init(p); /* Must remain after dup_task_struct() */
  copy_flags(clone_flags, p);
- p->pid = pid_nr(pid);
  INIT_LIST_HEAD(&p->children);
  INIT_LIST_HEAD(&p->sibling);
  p->vfork_done = NULL;
@@ -1112,10 +1106,6 @@ static struct task_struct *copy_process(
  p->blocked_on = NULL; /* not blocked yet */
 #endif

- p->tgid = p->pid;
- if (clone_flags & CLONE_THREAD)
-  p->tgid = current->tgid;
-
  if ((retval = security_task_alloc(p)))
   goto bad_fork_cleanup_policy;
  if ((retval = audit_alloc(p)))
@@ -1141,6 +1131,17 @@ static struct task_struct *copy_process(
  if (retval)
   goto bad_fork_cleanup_namespaces;

+ if (pid != &init_struct_pid) {
+  pid = alloc_pid(task_active_pid_ns(p));
+  if (!pid)
+   goto bad_fork_cleanup_namespaces;
+ }
+
```

```
+ p->pid = pid_nr(pid);
+ p->tgid = p->pid;
+ if (clone_flags & CLONE_THREAD)
+  p->tgid = current->tgid;
+
  p->set_child_tid = (clone_flags & CLONE_CHILD_SETTID) ? child_tidptr : NULL;
  /*
   * Clear TID on mm_release()?
@@ -1237,7 +1242,7 @@ static struct task_struct *copy_process(
   spin_unlock(&current->sighand->siglock);
   write_unlock_irq(&tasklist_lock);
   retval = -ERESTARTNOINTR;
-  goto bad_fork_cleanup_namespaces;
+  goto bad_fork_free_pid;
  }

  if (clone_flags & CLONE_THREAD) {
@@ -1266,11 +1271,22 @@ static struct task_struct *copy_process(
    __ptrace_link(p, current->parent);

   if (thread_group_leader(p)) {
-   p->signal->tty = current->signal->tty;
-   p->signal->pgrp = task_pgrp_nr(current);
-   set_task_session(p, task_session_nr(current));
-   attach_pid(p, PIDTYPE_PGID, task_pgrp(current));
-   attach_pid(p, PIDTYPE_SID, task_session(current));
+   if (clone_flags & CLONE_NEWPID) {
+    p->nsproxy->pid_ns->child_reaper = p;
+    p->signal->tty = NULL;
+    p->signal->pgrp = p->pid;
+    set_task_session(p, p->pid);
+    attach_pid(p, PIDTYPE_PGID, pid);
+    attach_pid(p, PIDTYPE_SID, pid);
+   } else {
+    p->signal->tty = current->signal->tty;
+    p->signal->pgrp = task_pgrp_nr(current);
+    set_task_session(p, task_session_nr(current));
+    attach_pid(p, PIDTYPE_PGID,
+      task_pgrp(current));
+    attach_pid(p, PIDTYPE_SID,
+      task_session(current));
+   }

    list_add_tail_rcu(&p->tasks, &init_task.tasks);
    __get_cpu_var(process_counts)++;
@@ -1288,6 +1304,9 @@ static struct task_struct *copy_process(
  container_post_fork(p);
  return p;
```

```
+bad_fork_free_pid:
+ if (pid != &init_struct_pid)
+  free_pid(pid);
 bad_fork_cleanup_namespaces:
  exit_task_namespaces(p);
 bad_fork_cleanup_keys:
@@ -1322,9 +1341,6 @@ bad_fork_cleanup_container:
 #endif
  container_exit(p, container_callbacks_done);
  delayacct_tsk_free(p);
- if (pid != &init_struct_pid)
-  free_pid(pid);
-bad_fork_put_binfmt_module:
  if (p->binfmt)
   module_put(p->binfmt->module);
 bad_fork_cleanup_put_domain:
@@ -1406,7 +1422,13 @@ long do_fork(unsigned long clone_flags,
  if (!IS_ERR(p)) {
   struct completion vfork;

-  nr = pid_nr(task_pid(p));
+  /*
+   * this is enough to call pid_nr_ns here, but this if
+   * improves optimisation of regular fork()
+   */
+  nr = (clone_flags & CLONE_NEWPID) ?
+   task_pid_nr_ns(p, current->nsproxy->pid_ns) :
+    task_pid_vnr(p);

   if (clone_flags & CLONE_VFORK) {
    p->vfork_done = &vfork;
```

Subject: [PATCH 10/15] Make each namespace has its own proc tree
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:54:22 GMT
View Forum Message <> Reply to Message

The idea is the following: when a namespace is created the
new proc superblock, pointing to this namespace, is created.
When user accesses the proc tree from some pountmoint it oberves
the tasks that live in the namespace, pointer by the according
spuperblock.

When task dies, its pids must be flushed from several proc trees,
i.e. the ones that are owned by the namespaces that can see the
taks's struct pid.

Having the namespaces live without the proc mount is racy, so
during the namespace creation we kern_mount the new proc instance.

We cannot just get the reference on the struct pid_namespace from
the superblock, otherwise we'll have the ns->proc_mnt->sb->ns
loop. To break this, when the init task dies, it releases the
kern_mount-ed instance.

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

```
 fs/proc/base.c              |   12 ++++++
 fs/proc/inode.c             |    3 +
 fs/proc/root.c              |   83 ++++++++++++++++++++++++++++++++++++++++++---
 include/linux/pid_namespace.h |    3 +
 include/linux/proc_fs.h     |   15 +++++++
 kernel/exit.c               |   18 ++++++++-
 kernel/fork.c               |    5 ++
 kernel/pid.c                |    1
 8 files changed, 133 insertions(+), 7 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/fs/proc/base.c linux-2.6.23-rc1-mm1-7/fs/proc/base.c
--- linux-2.6.23-rc1-mm1.orig/fs/proc/base.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/proc/base.c 2007-07-26 16:36:37.000000000 +0400
@@ -2161,7 +2162,19 @@ out:

 void proc_flush_task(struct task_struct *task, struct pid *pid)
 {
+ int i;
+ struct upid *upid;
+
  proc_flush_task_mnt(task, proc_mnt);
+ if (pid == NULL)
+  return;
+
+ for (i = 1; i <= pid->level; i++)
+  proc_flush_task_mnt(task, pid->numbers[i].ns->proc_mnt);
+
+ upid = &pid->numbers[pid->level];
+ if (upid->nr == 1)
+  pid_ns_release_proc(upid->ns);
 }

 static struct dentry *proc_pid_instantiate(struct inode *dir,
diff -upr linux-2.6.23-rc1-mm1.orig/fs/proc/inode.c linux-2.6.23-rc1-mm1-7/fs/proc/inode.c
--- linux-2.6.23-rc1-mm1.orig/fs/proc/inode.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/proc/inode.c 2007-07-26 16:36:36.000000000 +0400
```

```
@@ -16,6 +16,7 @@
 #include <linux/init.h>
 #include <linux/module.h>
 #include <linux/smp_lock.h>
+#include <linux/pid_namespace.h>

 #include <asm/system.h>
 #include <asm/uaccess.h>
@@ -427,7 +428,7 @@ out_mod:
  return NULL;
 }

-int proc_fill_super(struct super_block *s, void *data, int silent)
+int proc_fill_super(struct super_block *s)
 {
  struct inode * root_inode;

diff -upr linux-2.6.23-rc1-mm1.orig/fs/proc/root.c linux-2.6.23-rc1-mm1-7/fs/proc/root.c
--- linux-2.6.23-rc1-mm1.orig/fs/proc/root.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/proc/root.c 2007-07-26 16:36:37.000000000 +0400
@@ -18,32 +18,90 @@
 #include <linux/bitops.h>
 #include <linux/smp_lock.h>
 #include <linux/mount.h>
+#include <linux/pid_namespace.h>

 #include "internal.h"

 struct proc_dir_entry *proc_net, *proc_net_stat, *proc_bus, *proc_root_fs, *proc_root_driver;

+static int proc_test_super(struct super_block *sb, void *data)
+{
+ return sb->s_fs_info == data;
+}
+
+static int proc_set_super(struct super_block *sb, void *data)
+{
+ struct pid_namespace *ns;
+
+ ns = (struct pid_namespace *)data;
+ sb->s_fs_info = get_pid_ns(ns);
+ return set_anon_super(sb, NULL);
+}
+
 static int proc_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
 {
+ int err;
```

```
+ struct super_block *sb;
+ struct pid_namespace *ns;
+ struct proc_inode *ei;
+
  if (proc_mnt) {
   /* Seed the root directory with a pid so it doesn't need
    * to be special in base.c.  I would do this earlier but
    * the only task alive when /proc is mounted the first time
    * is the init_task and it doesn't have any pids.
    */
-  struct proc_inode *ei;
   ei = PROC_I(proc_mnt->mnt_sb->s_root->d_inode);
   if (!ei->pid)
    ei->pid = find_get_pid(1);
  }
- return get_sb_single(fs_type, flags, data, proc_fill_super, mnt);
+
+ if (flags & MS_KERNMOUNT)
+  ns = (struct pid_namespace *)data;
+ else
+  ns = current->nsproxy->pid_ns;
+
+ sb = sget(fs_type, proc_test_super, proc_set_super, ns);
+ if (IS_ERR(sb))
+  return PTR_ERR(sb);
+
+ if (!sb->s_root) {
+  sb->s_flags = flags;
+  err = proc_fill_super(sb);
+  if (err) {
+   up_write(&sb->s_umount);
+   deactivate_super(sb);
+   return err;
+  }
+
+  ei = PROC_I(sb->s_root->d_inode);
+  if (!ei->pid) {
+   rcu_read_lock();
+   ei->pid = get_pid(find_pid_ns(1, ns));
+   rcu_read_unlock();
+  }
+
+  sb->s_flags |= MS_ACTIVE;
+  ns->proc_mnt = mnt;
+ }
+
+ return simple_set_mnt(mnt, sb);
+}
```

```
+
+static void proc_kill_sb(struct super_block *sb)
+{
+ struct pid_namespace *ns;
+
+ ns = (struct pid_namespace *)sb->s_fs_info;
+ kill_anon_super(sb);
+ put_pid_ns(ns);
 }

 static struct file_system_type proc_fs_type = {
  .name  = "proc",
  .get_sb  = proc_get_sb,
- .kill_sb = kill_anon_super,
+ .kill_sb = proc_kill_sb,
 };

 void __init proc_root_init(void)
@@ -54,12 +112,13 @@ void __init proc_root_init(void)
  err = register_filesystem(&proc_fs_type);
  if (err)
   return;
- proc_mnt = kern_mount(&proc_fs_type);
+ proc_mnt = kern_mount_data(&proc_fs_type, &init_pid_ns);
  err = PTR_ERR(proc_mnt);
  if (IS_ERR(proc_mnt)) {
   unregister_filesystem(&proc_fs_type);
   return;
  }
+
  proc_misc_init();
  proc_net = proc_mkdir("net", NULL);
  proc_net_stat = proc_mkdir("net/stat", NULL);
@@ -153,6 +212,22 @@ struct proc_dir_entry proc_root = {
  .parent  = &proc_root,
 };

+int pid_ns_prepare_proc(struct pid_namespace *ns)
+{
+ struct vfsmount *mnt;
+
+ mnt = kern_mount_data(&proc_fs_type, ns);
+ if (IS_ERR(mnt))
+  return PTR_ERR(mnt);
+
+ return 0;
+}
+
```

```
+void pid_ns_release_proc(struct pid_namespace *ns)
+{
+ mntput(ns->proc_mnt);
+}
+
 EXPORT_SYMBOL(proc_symlink);
 EXPORT_SYMBOL(proc_mkdir);
 EXPORT_SYMBOL(create_proc_entry);
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h
linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h 2007-07-26 16:34:45.000000000
+0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h 2007-07-26 16:36:36.000000000
+0400
@@ -16,6 +15,9 @@ struct pidmap {
  int last_pid;
  struct task_struct *child_reaper;
  struct kmem_cache *pid_cachep;
+#ifdef CONFIG_PROC_FS
+ struct vfsmount *proc_mnt;
+#endif
 };

 extern struct pid_namespace init_pid_ns;
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/proc_fs.h
linux-2.6.23-rc1-mm1-7/include/linux/proc_fs.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/proc_fs.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/proc_fs.h 2007-07-26 16:36:36.000000000 +0400
@@ -126,7 +126,8 @@ extern struct proc_dir_entry *create_pro
 extern void remove_proc_entry(const char *name, struct proc_dir_entry *parent);

 extern struct vfsmount *proc_mnt;
-extern int proc_fill_super(struct super_block *,void *,int);
+struct pid_namespace;
+extern int proc_fill_super(struct super_block *);
 extern struct inode *proc_get_inode(struct super_block *, unsigned int, struct proc_dir_entry *);

 /*
@@ -143,6 +144,9 @@ extern const struct file_operations proc
 extern const struct file_operations proc_kmsg_operations;
 extern const struct file_operations ppc_htab_operations;

+extern int pid_ns_prepare_proc(struct pid_namespace *ns);
+extern void pid_ns_release_proc(struct pid_namespace *ns);
+
 /*
  * proc_tty.c
  */
```

```
@@ -248,6 +254,15 @@ static inline void proc_tty_unregister_d

 extern struct proc_dir_entry proc_root;

+static inline int pid_ns_prepare_proc(struct pid_namespace *ns)
+{
+ return 0;
+}
+
+static inline void pid_ns_release_proc(struct pid_namespace *ns)
+{
+}
+
 #endif /* CONFIG_PROC_FS */

 #if !defined(CONFIG_PROC_KCORE)
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c linux-2.6.23-rc1-mm1-7/kernel/exit.c
--- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26 16:36:37.000000000 +0400
@@ -153,6 +153,7 @@ static void delayed_put_task_struct(stru

 void release_task(struct task_struct * p)
 {
+ struct pid *pid;
  struct task_struct *leader;
  int zap_leader;
 repeat:
@@ -160,6 +161,20 @@ repeat:
  write_lock_irq(&tasklist_lock);
  ptrace_unlink(p);
  BUG_ON(!list_empty(&p->ptrace_list) || !list_empty(&p->ptrace_children));
+ /*
+  * we have to keep this pid till proc_flush_task() to make
+  * it possible to flush all dentries holding it. pid will
+  * be put ibidem
+  *
+  * however if the pid belogs to init namespace only, we can
+  * optimize this out
+  */
+ pid = task_pid(p);
+ if (pid->level != 0)
+  get_pid(pid);
+ else
+  pid = NULL;
+
  __exit_signal(p);

  /*
```

```
@@ -184,7 +199,8 @@ repeat:
  }

  write_unlock_irq(&tasklist_lock);
- proc_flush_task(p, NULL);
+ proc_flush_task(p, pid);
+ put_pid(pid);
  release_thread(p);
  call_rcu(&p->rcu, delayed_put_task_struct);

diff -upr linux-2.6.23-rc1-mm1.orig/kernel/fork.c linux-2.6.23-rc1-mm1-7/kernel/fork.c
--- linux-2.6.23-rc1-mm1.orig/kernel/fork.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/fork.c 2007-07-26 16:36:37.000000000 +0400
@@ -50,6 +50,7 @@
 #include <linux/taskstats_kern.h>
 #include <linux/random.h>
 #include <linux/tty.h>
+#include <linux/proc_fs.h>

 #include <asm/pgtable.h>
 #include <asm/pgalloc.h>
@@ -1141,6 +1131,10 @@ static struct task_struct *copy_process(
   pid = alloc_pid(task_active_pid_ns(p));
   if (!pid)
    goto bad_fork_cleanup_namespaces;
+
+  if (clone_flags & CLONE_NEWPID)
+   if (pid_ns_prepare_proc(task_active_pid_ns(p)))
+    goto bad_fork_free_pid;
  }

  p->pid = pid_nr(pid);
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -28,6 +28,7 @@
 #include <linux/hash.h>
 #include <linux/pid_namespace.h>
 #include <linux/init_task.h>
+#include <linux/proc_fs.h>

 #define pid_hashfn(nr, ns) \
  hash_long((unsigned long)nr + (unsigned long)ns, pidhash_shift)
```

Subject: [PATCH 11/15] Signal semantics
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:55:03 GMT

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

With support for multiple pid namespaces, each pid namespace has a
separate child reaper and this process needs some special handling
of signals.

- The child reaper should appear like a normal process to other
  processes in its ancestor namespaces and so should be killable
  (or not) in the usual way.

    - The child reaper should receive, from processes in it's active
      and decendent namespaces, only those signals for which it has
      installed a signal handler.

- System-wide signals (eg: kill signum -1) from within a child namespace
  should only affect processes within that namespace and descendant
  namespaces. They should not be posted to processes in ancestor or
  sibling namespaces.

- If the sender of a signal does not have a pid_t in the receiver's
  namespace (eg: a process in init_pid_ns sends a signal to a process
  in a descendant namespace), the sender's pid should appear as 0
  in the signal's siginfo structure.

- Existing rules for SIGIO delivery still apply and a process can
  choose any other process in its namespace and descendant namespaces
  to receive the SIGIO signal.

  The following appears to be incorrect in the fcntl() man page for
  F_SETOWN.

> Sending a signal to the owner process (group) specified by
> F_SETOWN is subject to the same permissions checks as are
> described for kill(2), where the sending process is the one that
> employs F_SETOWN (but see BUGS below).

Current behavior is that the SIGIO signal is delivered on behalf of
the process that caused the event (eg: made data available on the
file) and not the process that called fcntl().

To implement the above requirements, we:

- Add a check in check_kill_permission() for a process within a
  namespace sending the fast-pathed, SIGKILL signal.

- We use a flag, SIGQUEUE_CINIT, to tell the container-init if

a signal posted to its queue is from a process within its own
namespace. The flag is set in send_signal() if a process attempts
to send a signal to its container-init.

The SIGQUEUE_CINIT flag is checked in collect_signal() - if
the flag is set, collect_signal() sets the KERN_SIGINFO_CINIT
flag in the kern_siginfo. The KERN_SIGINFO_CINIT flag indicates
that the sender is from within the namespace and the container-init
can choose to ignore the signal.

If the KERN_SIGINFO_CINIT flag is clear in get_signal_to_deliver(),
the signal originated from an ancestor namespace and so the
container-init honors the signal.

Note: We currently use two flags, SIGQUEUE_CINIT, KERN_SIGINFO_CINIT to
avoid modifying 'struct sigqueue'. If 'kern_siginfo' approach is
feasible, we could use 'kern_siginfo' in sigqueue and eliminate
SIGQUEUE_CINIT.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

---

include/linux/pid.h | 3 ++
include/linux/signal.h | 1
kernel/pid.c | 46 ++
kernel/signal.c | 63 ++-
4 files changed, 112 insertions(+), 1 deletion(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid.h linux-2.6.23-rc1-mm1-7/include/linux/pid.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
@@ -71,6 +77,9 @@ extern struct task_struct *FASTCALL(pid_
extern struct task_struct *FASTCALL(get_pid_task(struct pid *pid,
enum pid_type));

+extern int task_visible_in_pid_ns(struct task_struct *tsk,
+ struct pid_namespace *ns);
+extern int pid_ns_equal(struct task_struct *tsk);
extern struct pid *get_task_pid(struct task_struct *task, enum pid_type type);

/*
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/signal.h
linux-2.6.23-rc1-mm1-7/include/linux/signal.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/signal.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/signal.h 2007-07-26 16:36:37.000000000 +0400

```
@@ -20,6 +27,7 @@ struct sigqueue {

 /* flags values. */
 #define SIGQUEUE_PREALLOC 1
+#define SIGQUEUE_CINIT  2

 struct sigpending {
  struct list_head list;
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -318,6 +355,52 @@ struct task_struct * fastcall pid_task(s
 }

 /*
+ * Return TRUE if the task @p is visible in the pid namespace @ns
+ *
+ * Note: @p is visible in @ns if the active-pid-ns of @p is either equal to
+ *  @ns or is a descendant of @ns.
+ *
+ *  @p is not visible in @ns if active-pid-ns of @p is an ancestor of @ns.
+ *  Eg: Processes in init-pid-ns are not visible in child pid namespaces.
+ *  They should not receive any system-wide signals from a child-pid-
+ *  namespace for instance.
+ */
+int task_visible_in_pid_ns(struct task_struct *p, struct pid_namespace *ns)
+{
+ int i;
+ struct pid *pid = task_pid(p);
+
+ if (!pid)
+  return 0;
+
+ for (i = 0; i <= pid->level; i++) {
+  if (pid->numbers[i].ns == ns)
+   return 1;
+ }
+
+ return 0;
+}
+EXPORT_SYMBOL(task_visible_in_pid_ns);
+
+/*
+ * Return TRUE if the active pid namespace of @tsk is same as active
+ * pid namespace of 'current'
+ */
+
+static inline struct pid_namespace *pid_active_ns(struct pid *pid)
```

```
+{
+ if (pid == NULL)
+  return NULL;
+
+ return pid->numbers[pid->level].ns;
+}
+
+int pid_ns_equal(struct task_struct *tsk)
+{
+ return pid_active_ns(task_pid(tsk)) == pid_active_ns(task_pid(current));
+}
+
+/*
  * Must be called under rcu_read_lock() or with tasklist_lock read-held.
  */
 struct task_struct *find_task_by_pid_type_ns(int type, int nr,
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/signal.c linux-2.6.23-rc1-mm1-7/kernel/signal.c
--- linux-2.6.23-rc1-mm1.orig/kernel/signal.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/signal.c 2007-07-26 16:36:37.000000000 +0400
@@ -323,6 +325,9 @@ static int collect_signal(int sig, struc
  if (first) {
   list_del_init(&first->list);
   copy_siginfo(info, &first->info);
+  if (first->flags & SIGQUEUE_CINIT)
+   kinfo->flags |= KERN_SIGINFO_CINIT;
+
   __sigqueue_free(first);
   if (!still_pending)
    sigdelset(&list->signal, sig);
@@ -343,6 +348,8 @@ static int collect_signal(int sig, struc
 {
  int sig = next_signal(pending, mask);

+ kinfo->flags &= ~KERN_SIGINFO_CINIT;
+
  if (sig) {
   if (current->notifier) {
    if (sigismember(current->notifier_mask, sig)) {
@@ -522,6 +547,20 @@ static int rm_from_queue(unsigned long m
  return 1;
 }

+static int deny_signal_to_container_init(struct task_struct *tsk, int sig)
+{
+ /*
+  * If receiver is the container-init of sender and signal is SIGKILL
+  * reject it right-away. If signal is any other one, let the container
+  * init decide (in get_signal_to_deliver()) whether to handle it or
```

```
+  * ignore it.
+  */
+ if (is_container_init(tsk) && (sig == SIGKILL) && pid_ns_equal(tsk))
+  return -EPERM;
+
+ return 0;
+}
+
 /*
  * Bad permissions for sending the signal
  */
@@ -545,6 +584,10 @@ static int check_kill_permission(int sig
      && !capable(CAP_KILL))
   return error;

+ error = deny_signal_to_container_init(t, sig);
+ if (error)
+  return error;
+
  return security_task_kill(t, info, sig, 0);
 }

@@ -659,6 +702,34 @@ static void handle_stop_signal(int sig,
  }
 }

+static void encode_sender_info(struct task_struct *t, struct sigqueue *q)
+{
+ /*
+  * If sender (i.e 'current') and receiver have the same active
+  * pid namespace and the receiver is the container-init, set the
+  * SIGQUEUE_CINIT flag. This tells the container-init that the
+  * signal originated in its own namespace and so it can choose
+  * to ignore the signal.
+  *
+  * If the receiver is the container-init of a pid namespace,
+  * but the sender is from an ancestor pid namespace, the
+  * container-init cannot ignore the signal. So clear the
+  * SIGQUEUE_CINIT flag in this case.
+  *
+  * Also, if the sender does not have a pid_t in the receiver's
+  * active pid namespace, set si_pid to 0 and pretend it originated
+  * from the kernel.
+  */
+ if (pid_ns_equal(t)) {
+  if (is_container_init(t)) {
+   q->flags |= SIGQUEUE_CINIT;
+  }
```

```
+ } else {
+  q->info.si_pid = 0;
+  q->info.si_code = SI_KERNEL;
+ }
+}
+
 static int send_signal(int sig, struct siginfo *info, struct task_struct *t,
    struct sigpending *signals)
 {
@@ -710,6 +781,7 @@ static int send_signal(int sig, struct s
    copy_siginfo(&q->info, info);
    break;
   }
+  encode_sender_info(t, q);
  } else if (!is_si_special(info)) {
   if (sig >= SIGRTMIN && info->si_code != SI_USER)
   /*
@@ -1149,6 +1221,8 @@ EXPORT_SYMBOL_GPL(kill_pid_info_as_uid);
 static int kill_something_info(int sig, struct siginfo *info, int pid)
 {
  int ret;
+ struct pid_namespace *my_ns = task_active_pid_ns(current);
+
  rcu_read_lock();
  if (!pid) {
   ret = kill_pgrp_info(sig, info, task_pgrp(current));
@@ -1158,6 +1232,13 @@ static int kill_something_info(int sig,

   read_lock(&tasklist_lock);
   for_each_process(p) {
+   /*
+    * System-wide signals apply only to the sender's
+    * pid namespace, unless issued from init_pid_ns.
+    */
+   if (!task_visible_in_pid_ns(p, my_ns))
+    continue;
+
    if (p->pid > 1 && p->tgid != current->tgid) {
     int err = group_send_sig_info(sig, info, p);
     ++count;
@@ -1852,7 +1950,7 @@ relock:
    * within that pid space. It can of course get signals from
    * its parent pid space.
    */
-  if (current == task_child_reaper(current))
+  if (kinfo.flags & KERN_SIGINFO_CINIT)
    continue;
```

if (sig_kernel_stop(signr)) {

Subject: [PATCH 12/15] Miscelaneous stuff for pid namespaces
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:56:01 GMT
View Forum Message <> Reply to Message

This includes
 * the headers dependency fix
 * task_child_reaper() correct declaration
 * fixes for is_global_init() and is_container_init()

Maybe this should come before all the other stuff...

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

---

 include/linux/pid_namespace.h | 4 ++--
 include/linux/sched.h | 4 ++--
 kernel/pid.c | 16 ++++++++++++---
 3 files changed, 17 insertions(+), 7 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h
linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h 2007-07-26 16:34:45.000000000
+0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h 2007-07-26 16:36:36.000000000
+0400
@@ -4,7 +4,6 @@
 #include <linux/sched.h>
 #include <linux/mm.h>
 #include <linux/threads.h>
-#include <linux/pid.h>
 #include <linux/nsproxy.h>
 #include <linux/kref.h>

@@ -46,7 +53,8 @@ static inline struct pid_namespace *task

 static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
 {
- return init_pid_ns.child_reaper;
+ BUG_ON(tsk != current);
+ return tsk->nsproxy->pid_ns->child_reaper;
 }

 #endif /* _LINUX_PID_NS_H */
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/sched.h

```
linux-2.6.23-rc1-mm1-7/include/linux/sched.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/sched.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/sched.h 2007-07-26 16:36:37.000000000 +0400
@@ -1277,13 +1366,13 @@ static inline int pid_alive(struct task_
  *
  * TODO: We should inline this function after some cleanups in pid_namespace.h
  */
-extern int is_global_init(struct task_struct *tsk);
+extern int is_container_init(struct task_struct *tsk);

 /*
  * is_container_init:
  * check whether in the task is init in its own pid namespace.
  */
-static inline int is_container_init(struct task_struct *tsk)
+static inline int is_global_init(struct task_struct *tsk)
 {
  return tsk->pid == 1;
 }
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -60,11 +62,21 @@ static inline int mk_pid(struct pid_name
 };
 EXPORT_SYMBOL(init_pid_ns);

-int is_global_init(struct task_struct *tsk)
+int is_container_init(struct task_struct *tsk)
 {
- return tsk == init_pid_ns.child_reaper;
+ int ret;
+ struct pid *pid;
+
+ ret = 0;
+ rcu_read_lock();
+ pid = task_pid(tsk);
+ if (pid != NULL && pid->numbers[pid->level].nr == 1)
+  ret = 1;
+ rcu_read_unlock();
+
+ return ret;
 }
-EXPORT_SYMBOL(is_global_init);
+EXPORT_SYMBOL(is_container_init);

 /*
  * Note: disable interrupts while the pidmap_lock is held as an
```

Subject: [PATCH 13/15] Clone the pid namespace
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:56:50 GMT
View Forum Message <> Reply to Message

When clone() is invoked with CLONE_NEWPID, create a new pid namespace
Since the active pid namespace is special and expected to be the first
entry in pid->upid_list, preserve the order of pid namespaces.

Pid namespaces can be nested and the nesting depth is unlimited.
When a process clones its pid namespace, we create additional pid caches
as necessary and use the pid cache to allocate 'struct pids' for that depth.

TODO:
 One of the reasons the free_work() was introduced was to cleanup
 the proc in non-atomic context, but since proc is now released
 from proc_flush_task() this looks to be unneeded, but I have to
 recheck this.

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

---

 include/linux/pid_namespace.h | 7 ++
 include/linux/sched.h | 1
 kernel/nsproxy.c | 3 -
 kernel/pid.c | 102 +++++++++++++++++++++++++++++++++++-----
 4 files changed, 101 insertions(+), 12 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h
linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h 2007-07-26 16:34:45.000000000
+0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h 2007-07-26 16:36:36.000000000
+0400
@@ -16,11 +15,16 @@ struct pidmap {
 #define PIDMAP_ENTRIES   ((PID_MAX_LIMIT + 8*PAGE_SIZE - 1)/PAGE_SIZE/8)

 struct pid_namespace {
- struct kref kref;
+ union {
+ struct kref kref;
+ struct work_struct free_work;
+ };
  struct pidmap pidmap[PIDMAP_ENTRIES];
  int last_pid;
  struct task_struct *child_reaper;
  struct kmem_cache *pid_cachep;
+ int level;
+ struct pid_namespace *parent;

```
 #ifdef CONFIG_PROC_FS
  struct vfsmount *proc_mnt;
 #endif
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/sched.h
linux-2.6.23-rc1-mm1-7/include/linux/sched.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/sched.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/sched.h 2007-07-26 16:36:37.000000000 +0400
@@ -27,6 +27,7 @@
 #define CLONE_NEWUTS  0x04000000 /* New utsname group? */
 #define CLONE_NEWIPC  0x08000000 /* New ipcs */
 #define CLONE_NEWUSER  0x10000000 /* New user namespace */
+#define CLONE_NEWPID  0x20000000 /* New pids */

 /*
  * Scheduling policies
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/nsproxy.c linux-2.6.23-rc1-mm1-7/kernel/nsproxy.c
--- linux-2.6.23-rc1-mm1.orig/kernel/nsproxy.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/nsproxy.c 2007-07-26 16:36:36.000000000 +0400
@@ -132,7 +132,8 @@ int copy_namespaces(unsigned long flags,

  get_nsproxy(old_ns);

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

  if (!capable(CAP_SYS_ADMIN)) {
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -60,14 +62,17 @@ static inline int mk_pid(struct pid_name
  * the scheme scales to up to 4 million PIDs, runtime.
  */
 struct pid_namespace init_pid_ns = {
- .kref = {
-  .refcount       = ATOMIC_INIT(2),
+ {
+  .kref = {
+   .refcount       = ATOMIC_INIT(2),
+  },
  },
  .pidmap = {
   [ 0 ... PIDMAP_ENTRIES-1] = { ATOMIC_INIT(BITS_PER_PAGE), NULL }
  },
  .last_pid = 0,
- .child_reaper = &init_task
+ .level = 0,
```

```
+ .child_reaper = &init_task,
 };
 EXPORT_SYMBOL(init_pid_ns);

@@ -409,8 +501,8 @@ static struct kmem_cache *create_pid_cac

  snprintf(pcache->name, sizeof(pcache->name), "pid_%d", nr_ids);
  cachep = kmem_cache_create(pcache->name,
-   /* FIXME add numerical ids here */
-   sizeof(struct pid), 0, SLAB_HWCACHE_ALIGN, NULL);
+   sizeof(struct pid) + (nr_ids - 1) * sizeof(struct upid),
+   0, SLAB_HWCACHE_ALIGN, NULL);
  if (cachep == NULL)
   goto err_cachep;

@@ -428,11 +520,89 @@ err_alloc:
  return NULL;
 }

-struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
+static struct pid_namespace *create_pid_namespace(int level)
+{
+ struct pid_namespace *ns;
+ int i;
+
+ ns = kmalloc(sizeof(struct pid_namespace), GFP_KERNEL);
+ if (ns == NULL)
+  goto out;
+
+ ns->pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
+ if (!ns->pidmap[0].page)
+  goto out_free;
+
+ ns->pid_cachep = create_pid_cachep(level + 1);
+ if (ns->pid_cachep == NULL)
+  goto out_free_map;
+
+ kref_init(&ns->kref);
+ ns->last_pid = 0;
+ ns->child_reaper = NULL;
+ ns->level = level;
+
+ set_bit(0, ns->pidmap[0].page);
+ atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
+ get_pid_ns(ns);
+
+ for (i = 1; i < PIDMAP_ENTRIES; i++) {
+  ns->pidmap[i].page = 0;
```

```
+  atomic_set(&ns->pidmap[i].nr_free, BITS_PER_PAGE);
+ }
+
+ return ns;
+
+out_free_map:
+ kfree(ns->pidmap[0].page);
+out_free:
+ kfree(ns);
+out:
+ return ERR_PTR(-ENOMEM);
+}
+
+static void destroy_pid_namespace(struct pid_namespace *ns)
 {
- BUG_ON(!old_ns);
- get_pid_ns(old_ns);
- return old_ns;
+ int i;
+
+ for (i = 0; i < PIDMAP_ENTRIES; i++)
+  kfree(ns->pidmap[i].page);
+ kfree(ns);
+}
+
+  struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
+  {
+ struct pid_namespace *new_ns;
+
+   BUG_ON(!old_ns);
+ new_ns = get_pid_ns(old_ns);
+ if (!(flags & CLONE_NEWPID))
+  goto out;
+
+ new_ns = ERR_PTR(-EINVAL);
+ if (flags & CLONE_THREAD)
+  goto out_put;
+
+ new_ns = create_pid_namespace(old_ns->level + 1);
+ if (new_ns != NULL)
+  new_ns->parent = get_pid_ns(old_ns);
+
+out_put:
+ put_pid_ns(old_ns);
+out:
+ return new_ns;
+}
+
```

```
+static void do_free_pid_ns(struct work_struct *w)
+{
+ struct pid_namespace *ns, *parent;
+
+ ns = container_of(w, struct pid_namespace, free_work);
+ parent = ns->parent;
+ destroy_pid_namespace(ns);
+
+ if (parent != NULL)
+  put_pid_ns(parent);
 }

 void free_pid_ns(struct kref *kref)
@@ -440,7 +648,8 @@ void free_pid_ns(struct kref *kref)
  struct pid_namespace *ns;

  ns = container_of(kref, struct pid_namespace, kref);
- kfree(ns);
+ INIT_WORK(&ns->free_work, do_free_pid_ns);
+ schedule_work(&ns->free_work);
 }

 /*
```

Subject: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:57:39 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

Terminate all processes in a namespace when the reaper of the namespace
is exiting. We do this by walking the pidmap of the namespace and sending
SIGKILL to all processes.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Acked-by: Pavel Emelyanov <xemul@openvz.org>

```
 include/linux/pid.h  |    1 +
 include/linux/wait.h |    4 ++++
 kernel/exit.c        |   10 ++++++----
 kernel/pid.c         |   38 ++++++++++++++++++++++++++++++++++++++
 4 files changed, 49 insertions(+), 4 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid.h linux-2.6.23-rc1-mm1-7/include/linux/pid.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
```

```
+++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
@@ -83,6 +92,7 @@ extern void FASTCALL(detach_pid(struct t

 extern struct pid *alloc_pid(struct pid_namespace *ns);
 extern void FASTCALL(free_pid(struct pid *pid));
+extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

 /*
  * the helpers to get the pid's id seen from different namespaces
diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/wait.h linux-2.6.23-rc1-mm1-7/include/linux/wait.h
--- linux-2.6.23-rc1-mm1.orig/include/linux/wait.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/linux/wait.h 2007-07-26 16:36:36.000000000 +0400
@@ -22,9 +22,13 @@
 #include <linux/list.h>
 #include <linux/stddef.h>
 #include <linux/spinlock.h>
+#include <linux/resource.h>
+#include <asm/siginfo.h>
 #include <asm/system.h>
 #include <asm/current.h>

+long do_wait(pid_t pid, int options, struct siginfo __user *infop,
+  int __user *stat_addr, struct rusage __user *ru);
 typedef struct __wait_queue wait_queue_t;
 typedef int (*wait_queue_func_t)(wait_queue_t *wait, unsigned mode, int sync, void *key);
 int default_wake_function(wait_queue_t *wait, unsigned mode, int sync, void *key);
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c linux-2.6.23-rc1-mm1-7/kernel/exit.c
--- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26 16:36:37.000000000 +0400
@@ -895,6 +915,7 @@ fastcall NORET_TYPE void do_exit(long co
 {
  struct task_struct *tsk = current;
  int group_dead;
+ struct pid_namespace *pid_ns = tsk->nsproxy->pid_ns;

  profile_task_exit(tsk);

@@ -905,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
  if (unlikely(!tsk->pid))
   panic("Attempted to kill the idle task!");
  if (unlikely(tsk == task_child_reaper(tsk))) {
-  if (task_active_pid_ns(tsk) != &init_pid_ns)
-   task_active_pid_ns(tsk)->child_reaper =
-    init_pid_ns.child_reaper;
+  if (pid_ns != &init_pid_ns) {
+   zap_pid_ns_processes(pid_ns);
+   pid_ns->child_reaper = init_pid_ns.child_reaper;
+  }
```

```
   else
    panic("Attempted to kill init!");
  }
@@ -1518,7 +1548,7 @@ static inline int my_ptrace_child(struct
  return (p->parent != p->real_parent);
 }

-static long do_wait(pid_t pid, int options, struct siginfo __user *infop,
+long do_wait(pid_t pid, int options, struct siginfo __user *infop,
       int __user *stat_addr, struct rusage __user *ru)
 {
  DECLARE_WAITQUEUE(wait, current);
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
--- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
@@ -428,6 +520,44 @@ err_alloc:
  return new_ns;
 }

+void zap_pid_ns_processes(struct pid_namespace *pid_ns)
+{
+ int i;
+ int nr;
+ int nfree;
+ int options = WNOHANG|WEXITED|__WALL;
+
+repeat:
+ /*
+  * We know pid == 1 is terminating. Find remaining pid_ts
+  * in the namespace, signal them and then wait for them
+  * exit.
+  */
+ nr = next_pidmap(pid_ns, 1);
+ while (nr > 0) {
+  kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
+  nr = next_pidmap(pid_ns, nr);
+ }
+
+ nr = next_pidmap(pid_ns, 1);
+ while (nr > 0) {
+  do_wait(nr, options, NULL, NULL, NULL);
+  nr = next_pidmap(pid_ns, nr);
+ }
+
+ nfree = 0;
+ for (i = 0; i < PIDMAP_ENTRIES; i++)
+  nfree += atomic_read(&pid_ns->pidmap[i].nr_free);
+
```

```
+ /*
+  * If pidmap has entries for processes other than 0 and 1, retry.
+  */
+ if (nfree < (BITS_PER_PAGE * PIDMAP_ENTRIES - 2))
+  goto repeat;
+
+ return;
+}
+
 static void do_free_pid_ns(struct work_struct *w)
 {
  struct pid_namespace *ns, *parent;
```

Subject: [PATCH 15/15] Hooks over the code to show correct values to user
Posted by Pavel Emelianov on Thu, 26 Jul 2007 14:58:53 GMT
View Forum Message <> Reply to Message

This is the largest patch in the set. Make all (I hope) the places where
the pid is shown to or get from user operate on the virtual pids.

The idea is:
 - all in-kernel data structures must store either struct pid itself
   or the pid's global nr, obtained with pid_nr() call;
 - when seeking the task from kernel code with the stored id one
   should use find_task_by_pid() call that works with global pids;
 - when showing pid's numerical value to the user the virtual one
   should be used, but however when one shows task's pid outside this
   task's namespace the global one is to be used;
 - when getting the pid from userspace one need to consider this as
   the virtual one and use appropriate task/pid-searching functions.

Many thanks to Sukadev for pointing out many places I lost.

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

```
 arch/ia64/kernel/signal.c        |    4 +--
 arch/parisc/kernel/signal.c      |    2 -
 arch/sparc/kernel/sys_sunos.c    |    2 -
 arch/sparc64/kernel/sys_sunos32.c |    2 -
 drivers/char/tty_io.c            |    7 +++---
 fs/binfmt_elf.c                  |   16 +++++++-------
 fs/binfmt_elf_fdpic.c            |   16 +++++++-------
 fs/exec.c                        |    4 +--
 fs/fcntl.c                       |    5 ++--
 fs/ioprio.c                      |    5 ++--
```

```
 fs/proc/array.c               |   25 +++++++++++++++++++-------
 fs/proc/base.c                |   37 ++++++++++++++++++++++++++-----------
 include/net/scm.h             |    4 ++-
 ipc/mqueue.c                  |    7 ++++--
 ipc/msg.c                     |    6 ++---
 ipc/sem.c                     |    8 +++----
 ipc/shm.c                     |    6 ++---
 kernel/capability.c           |   14 +++++++-----
 kernel/exit.c                 |   31 +++++++++++++++++++++++++++++++
 kernel/fork.c                 |    2 -
 kernel/futex.c                |   20 +++++++++++++++++++
 kernel/ptrace.c               |    4 ++-
 kernel/sched.c                |    3 +-
 kernel/signal.c               |   39 +++++++++++++++++++++++++++++++++++++
 kernel/sys.c                  |   42 +++++++++++++++++++++++++++++++++++++++++
 kernel/sysctl.c               |    2 -
 kernel/timer.c                |    7 +++---
 net/core/scm.c                |    4 ++-
 net/unix/af_unix.c            |    6 ++---
 29 files changed, 198 insertions(+), 132 deletions(-)

diff -upr linux-2.6.23-rc1-mm1.orig/arch/ia64/kernel/signal.c
linux-2.6.23-rc1-mm1-7/arch/ia64/kernel/signal.c
--- linux-2.6.23-rc1-mm1.orig/arch/ia64/kernel/signal.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/arch/ia64/kernel/signal.c 2007-07-26 16:36:36.000000000 +0400
@@ -227,7 +227,7 @@ ia64_rt_sigreturn (struct sigscratch *sc
  si.si_signo = SIGSEGV;
  si.si_errno = 0;
  si.si_code = SI_KERNEL;
- si.si_pid = current->pid;
+ si.si_pid = task_pid_vnr(current);
  si.si_uid = current->uid;
  si.si_addr = sc;
  force_sig_info(SIGSEGV, &si, current);
@@ -332,7 +332,7 @@ force_sigsegv_info (int sig, void __user
  si.si_signo = SIGSEGV;
  si.si_errno = 0;
  si.si_code = SI_KERNEL;
- si.si_pid = current->pid;
+ si.si_pid = task_pid_vnr(current);
  si.si_uid = current->uid;
  si.si_addr = addr;
  force_sig_info(SIGSEGV, &si, current);
diff -upr linux-2.6.23-rc1-mm1.orig/arch/parisc/kernel/signal.c
linux-2.6.23-rc1-mm1-7/arch/parisc/kernel/signal.c
--- linux-2.6.23-rc1-mm1.orig/arch/parisc/kernel/signal.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/arch/parisc/kernel/signal.c 2007-07-26 16:36:36.000000000 +0400
@@ -181,7 +181,7 @@ give_sigsegv:
```

```
  si.si_signo = SIGSEGV;
  si.si_errno = 0;
  si.si_code = SI_KERNEL;
- si.si_pid = current->pid;
+ si.si_pid = task_pid_vnr(current);
  si.si_uid = current->uid;
  si.si_addr = &frame->uc;
  force_sig_info(SIGSEGV, &si, current);
diff -upr linux-2.6.23-rc1-mm1.orig/arch/sparc/kernel/sys_sunos.c
linux-2.6.23-rc1-mm1-7/arch/sparc/kernel/sys_sunos.c
--- linux-2.6.23-rc1-mm1.orig/arch/sparc/kernel/sys_sunos.c 2007-07-26 16:34:45.000000000
+0400
+++ linux-2.6.23-rc1-mm1-7/arch/sparc/kernel/sys_sunos.c 2007-07-26 16:36:36.000000000
+0400
@@ -866,7 +866,7 @@ asmlinkage int sunos_killpg(int pgrp, in
  rcu_read_lock();
  ret = -EINVAL;
  if (pgrp > 0)
-  ret = kill_pgrp(find_pid(pgrp), sig, 0);
+  ret = kill_pgrp(find_vpid(pgrp), sig, 0);
  rcu_read_unlock();

  return ret;
diff -upr linux-2.6.23-rc1-mm1.orig/arch/sparc64/kernel/sys_sunos32.c
linux-2.6.23-rc1-mm1-7/arch/sparc64/kernel/sys_sunos32.c
--- linux-2.6.23-rc1-mm1.orig/arch/sparc64/kernel/sys_sunos32.c 2007-07-26
16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/arch/sparc64/kernel/sys_sunos32.c 2007-07-26 16:36:36.000000000
+0400
@@ -831,7 +831,7 @@ asmlinkage int sunos_killpg(int pgrp, in
  rcu_read_lock();
  ret = -EINVAL;
  if (pgrp > 0)
-  ret = kill_pgrp(find_pid(pgrp), sig, 0);
+  ret = kill_pgrp(find_vpid(pgrp), sig, 0);
  rcu_read_unlock();

  return ret;
diff -upr linux-2.6.23-rc1-mm1.orig/drivers/char/tty_io.c
linux-2.6.23-rc1-mm1-7/drivers/char/tty_io.c
--- linux-2.6.23-rc1-mm1.orig/drivers/char/tty_io.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/drivers/char/tty_io.c 2007-07-26 16:36:36.000000000 +0400
@@ -103,6 +103,7 @@
 #include <linux/selection.h>

 #include <linux/kmod.h>
+#include <linux/nsproxy.h>
```

```
 #undef TTY_DEBUG_HANGUP

@@ -3080,7 +3081,7 @@ static int tiocgpgrp(struct tty_struct *
    */
   if (tty == real_tty && current->signal->tty != real_tty)
    return -ENOTTY;
- return put_user(pid_nr(real_tty->pgrp), p);
+ return put_user(pid_vnr(real_tty->pgrp), p);
 }

 /**
@@ -3114,7 +3115,7 @@ static int tiocspgrp(struct tty_struct *
   if (pgrp_nr < 0)
    return -EINVAL;
   rcu_read_lock();
- pgrp = find_pid(pgrp_nr);
+ pgrp = find_vpid(pgrp_nr);
   retval = -ESRCH;
   if (!pgrp)
    goto out_unlock;
@@ -3151,7 +3152,7 @@ static int tiocgsid(struct tty_struct *t
    return -ENOTTY;
   if (!real_tty->session)
    return -ENOTTY;
- return put_user(pid_nr(real_tty->session), p);
+ return put_user(pid_vnr(real_tty->session), p);
 }

 /**
diff -upr linux-2.6.23-rc1-mm1.orig/fs/binfmt_elf.c linux-2.6.23-rc1-mm1-7/fs/binfmt_elf.c
--- linux-2.6.23-rc1-mm1.orig/fs/binfmt_elf.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/binfmt_elf.c 2007-07-26 16:36:36.000000000 +0400
@@ -1339,10 +1339,10 @@ static void fill_prstatus(struct elf_prs
   prstatus->pr_info.si_signo = prstatus->pr_cursig = signr;
   prstatus->pr_sigpend = p->pending.signal.sig[0];
   prstatus->pr_sighold = p->blocked.sig[0];
- prstatus->pr_pid = p->pid;
- prstatus->pr_ppid = p->parent->pid;
- prstatus->pr_pgrp = task_pgrp_nr(p);
- prstatus->pr_sid = task_session_nr(p);
+ prstatus->pr_pid = task_pid_vnr(p);
+ prstatus->pr_ppid = task_pid_vnr(p->parent);
+ prstatus->pr_pgrp = task_pgrp_vnr(p);
+ prstatus->pr_sid = task_session_vnr(p);
   if (thread_group_leader(p)) {
    /*
     * This is the record for the group leader.  Add in the
@@ -1385,10 +1385,10 @@ static int fill_psinfo(struct elf_prpsin
```

```
    psinfo->pr_psargs[i] = ' ';
  psinfo->pr_psargs[len] = 0;

- psinfo->pr_pid = p->pid;
- psinfo->pr_ppid = p->parent->pid;
- psinfo->pr_pgrp = task_pgrp_nr(p);
- psinfo->pr_sid = task_session_nr(p);
+ psinfo->pr_pid = task_pid_vnr(p);
+ psinfo->pr_ppid = task_pid_vnr(p->parent);
+ psinfo->pr_pgrp = task_pgrp_vnr(p);
+ psinfo->pr_sid = task_session_nr(p);

  i = p->state ? ffz(~p->state) + 1 : 0;
  psinfo->pr_state = i;
diff -upr linux-2.6.23-rc1-mm1.orig/fs/binfmt_elf_fdpic.c
linux-2.6.23-rc1-mm1-7/fs/binfmt_elf_fdpic.c
--- linux-2.6.23-rc1-mm1.orig/fs/binfmt_elf_fdpic.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/binfmt_elf_fdpic.c 2007-07-26 16:36:36.000000000 +0400
@@ -1342,10 +1342,10 @@ static void fill_prstatus(struct elf_prs
  prstatus->pr_info.si_signo = prstatus->pr_cursig = signr;
  prstatus->pr_sigpend = p->pending.signal.sig[0];
  prstatus->pr_sighold = p->blocked.sig[0];
- prstatus->pr_pid = p->pid;
- prstatus->pr_ppid = p->parent->pid;
- prstatus->pr_pgrp = task_pgrp_nr(p);
- prstatus->pr_sid = task_session_nr(p);
+ prstatus->pr_pid = task_pid_vnr(p);
+ prstatus->pr_ppid = task_pid_vnr(p->parent);
+ prstatus->pr_pgrp = task_pgrp_vnr(p);
+ prstatus->pr_sid = task_session_nr(p);
  if (thread_group_leader(p)) {
   /*
    * This is the record for the group leader.  Add in the
@@ -1391,10 +1391,10 @@ static int fill_psinfo(struct elf_prpsin
    psinfo->pr_psargs[i] = ' ';
  psinfo->pr_psargs[len] = 0;

- psinfo->pr_pid = p->pid;
- psinfo->pr_ppid = p->parent->pid;
- psinfo->pr_pgrp = task_pgrp_nr(p);
- psinfo->pr_sid = task_session_nr(p);
+ psinfo->pr_pid = task_pid_vnr(p);
+ psinfo->pr_ppid = task_pid_vnr(p->parent);
+ psinfo->pr_pgrp = task_pgrp_vnr(p);
+ psinfo->pr_sid = task_session_nr(p);

  i = p->state ? ffz(~p->state) + 1 : 0;
  psinfo->pr_state = i;
```

```
diff -upr linux-2.6.23-rc1-mm1.orig/fs/exec.c linux-2.6.23-rc1-mm1-7/fs/exec.c
--- linux-2.6.23-rc1-mm1.orig/fs/exec.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/exec.c 2007-07-26 16:36:36.000000000 +0400
@@ -1462,7 +1462,7 @@ static int format_corename(char *corenam
    case 'p':
     pid_in_pattern = 1;
     rc = snprintf(out_ptr, out_end - out_ptr,
-           "%d", current->tgid);
+           "%d", task_tgid_vnr(current));
     if (rc > out_end - out_ptr)
      goto out;
     out_ptr += rc;
@@ -1534,7 +1534,7 @@ static int format_corename(char *corenam
  if (!ispipe && !pid_in_pattern
            && (core_uses_pid || atomic_read(&current->mm->mm_users) != 1)) {
   rc = snprintf(out_ptr, out_end - out_ptr,
-         ".%d", current->tgid);
+         ".%d", task_tgid_vnr(current));
   if (rc > out_end - out_ptr)
    goto out;
   out_ptr += rc;
diff -upr linux-2.6.23-rc1-mm1.orig/fs/fcntl.c linux-2.6.23-rc1-mm1-7/fs/fcntl.c
--- linux-2.6.23-rc1-mm1.orig/fs/fcntl.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/fcntl.c 2007-07-26 16:36:36.000000000 +0400
@@ -18,6 +18,7 @@
 #include <linux/ptrace.h>
 #include <linux/signal.h>
 #include <linux/rcupdate.h>
+#include <linux/pid_namespace.h>

 #include <asm/poll.h>
 #include <asm/siginfo.h>
@@ -289,7 +290,7 @@ int f_setown(struct file *filp, unsigned
   who = -who;
  }
  rcu_read_lock();
- pid = find_pid(who);
+ pid = find_vpid(who);
  result = __f_setown(filp, pid, type, force);
  rcu_read_unlock();
  return result;
@@ -305,7 +306,7 @@ pid_t f_getown(struct file *filp)
 {
  pid_t pid;
  read_lock(&filp->f_owner.lock);
- pid = pid_nr(filp->f_owner.pid);
+ pid = pid_nr_ns(filp->f_owner.pid, current->nsproxy->pid_ns);
  if (filp->f_owner.pid_type == PIDTYPE_PGID)
```

```
   pid = -pid;
  read_unlock(&filp->f_owner.lock);
diff -upr linux-2.6.23-rc1-mm1.orig/fs/ioprio.c linux-2.6.23-rc1-mm1-7/fs/ioprio.c
--- linux-2.6.23-rc1-mm1.orig/fs/ioprio.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/ioprio.c 2007-07-26 16:36:36.000000000 +0400
@@ -25,6 +25,7 @@
 #include <linux/capability.h>
 #include <linux/syscalls.h>
 #include <linux/security.h>
+#include <linux/pid_namespace.h>

 static int set_task_ioprio(struct task_struct *task, int ioprio)
 {
@@ -101,7 +102,7 @@ asmlinkage long sys_ioprio_set(int which
    if (!who)
     pgrp = task_pgrp(current);
    else
-    pgrp = find_pid(who);
+    pgrp = find_vpid(who);
    do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
     ret = set_task_ioprio(p, ioprio);
     if (ret)
@@ -188,7 +189,7 @@ asmlinkage long sys_ioprio_get(int which
    if (!who)
     pgrp = task_pgrp(current);
    else
-    pgrp = find_pid(who);
+    pgrp = find_vpid(who);
    do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
     tmpio = get_task_ioprio(p);
     if (tmpio < 0)
diff -upr linux-2.6.23-rc1-mm1.orig/fs/proc/array.c linux-2.6.23-rc1-mm1-7/fs/proc/array.c
--- linux-2.6.23-rc1-mm1.orig/fs/proc/array.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/proc/array.c 2007-07-26 16:36:36.000000000 +0400
@@ -77,6 +77,7 @@
 #include <linux/cpuset.h>
 #include <linux/rcupdate.h>
 #include <linux/delayacct.h>
+#include <linux/pid_namespace.h>

 #include <asm/pgtable.h>
 #include <asm/processor.h>
@@ -161,7 +162,9 @@ static inline char *task_state(struct ta
  struct group_info *group_info;
  int g;
  struct fdtable *fdt = NULL;
+ struct pid_namespace *ns;
```

```
+ ns = current->nsproxy->pid_ns;
  rcu_read_lock();
  buffer += sprintf(buffer,
   "State:\t%s\n"
@@ -172,9 +175,12 @@ static inline char *task_state(struct ta
   "Uid:\t%d\t%d\t%d\t%d\n"
   "Gid:\t%d\t%d\t%d\t%d\n",
   get_task_state(p),
-  p->tgid, p->pid,
-  pid_alive(p) ? rcu_dereference(p->real_parent)->tgid : 0,
-  pid_alive(p) && p->ptrace ? rcu_dereference(p->parent)->pid : 0,
+         task_tgid_nr_ns(p, ns),
+  task_pid_nr_ns(p, ns),
+         pid_alive(p) ?
+   task_ppid_nr_ns(p, ns) : 0,
+  pid_alive(p) && p->ptrace ?
+   task_tgid_nr_ns(rcu_dereference(p->parent), ns) : 0,
   p->uid, p->euid, p->suid, p->fsuid,
   p->gid, p->egid, p->sgid, p->fsgid);

@@ -373,6 +379,9 @@ static int do_task_stat(struct task_stru
  unsigned long rsslim = 0;
  char tcomm[sizeof(task->comm)];
  unsigned long flags;
+ struct pid_namespace *ns;
+
+ ns = current->nsproxy->pid_ns;

  state = *get_task_state(task);
  vsize = eip = esp = 0;
@@ -395,7 +404,7 @@ static int do_task_stat(struct task_stru
   struct signal_struct *sig = task->signal;

   if (sig->tty) {
-   tty_pgrp = pid_nr(sig->tty->pgrp);
+   tty_pgrp = pid_nr_ns(sig->tty->pgrp, ns);
    tty_nr = new_encode_dev(tty_devnum(sig->tty));
   }

@@ -425,9 +434,9 @@ static int do_task_stat(struct task_stru
    stime += cputime_to_clock_t(sig->stime);
   }

-  sid = task_session_nr(task);
-  pgid = task_pgrp_nr(task);
-  ppid = rcu_dereference(task->real_parent)->tgid;
+  sid = task_session_nr_ns(task, ns);
+  pgid = task_pgrp_nr_ns(task, ns);
```

```
+  ppid = task_ppid_nr_ns(task, ns);

   unlock_task_sighand(task, &flags);
  }
@@ -458,7 +467,7 @@ static int do_task_stat(struct task_stru
  res = sprintf(buffer, "%d (%s) %c %d %d %d %d %d %u %lu \
 %lu %lu %lu %lu %lu %ld %ld %ld %ld %d 0 %llu %lu %ld %lu %lu %lu %lu %lu \
 %lu %lu %lu %lu %lu %lu %lu %lu %d %d %u %u %llu\n",
-  task->pid,
+  task_pid_nr_ns(task, ns),
   tcomm,
   state,
   ppid,
diff -upr linux-2.6.23-rc1-mm1.orig/fs/proc/base.c linux-2.6.23-rc1-mm1-7/fs/proc/base.c
--- linux-2.6.23-rc1-mm1.orig/fs/proc/base.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/fs/proc/base.c 2007-07-26 16:36:37.000000000 +0400
@@ -1844,14 +1845,14 @@ static int proc_self_readlink(struct den
         int buflen)
 {
  char tmp[PROC_NUMBUF];
- sprintf(tmp, "%d", current->tgid);
+ sprintf(tmp, "%d", task_tgid_vnr(current));
  return vfs_readlink(dentry,buffer,buflen,tmp);
 }

 static void *proc_self_follow_link(struct dentry *dentry, struct nameidata *nd)
 {
  char tmp[PROC_NUMBUF];
- sprintf(tmp, "%d", current->tgid);
+ sprintf(tmp, "%d", task_tgid_vnr(current));
  return ERR_PTR(vfs_follow_link(nd,tmp));
 }

@@ -2196,6 +2220,7 @@ struct dentry *proc_pid_lookup(struct in
  struct dentry *result = ERR_PTR(-ENOENT);
  struct task_struct *task;
  unsigned tgid;
+ struct pid_namespace *ns;

  result = proc_base_lookup(dir, dentry);
  if (!IS_ERR(result) || PTR_ERR(result) != -ENOENT)
@@ -2205,8 +2230,9 @@ struct dentry *proc_pid_lookup(struct in
  if (tgid == ~0U)
   goto out;

+ ns = (struct pid_namespace *)dentry->d_sb->s_fs_info;
  rcu_read_lock();
- task = find_task_by_pid(tgid);
```

```
+ task = find_task_by_pid_ns(tgid, ns);
  if (task)
   get_task_struct(task);
  rcu_read_unlock();
@@ -2223,7 +2249,8 @@ out:
  * Find the first task with tgid >= tgid
  *
  */
-static struct task_struct *next_tgid(unsigned int tgid)
+static struct task_struct *next_tgid(unsigned int tgid,
+  struct pid_namespace *ns)
 {
  struct task_struct *task;
  struct pid *pid;
@@ -2231,9 +2258,9 @@ static struct task_struct *next_tgid(uns
  rcu_read_lock();
 retry:
  task = NULL;
- pid = find_ge_pid(tgid);
+  pid = find_ge_pid(tgid, ns);
  if (pid) {
-  tgid = pid->nr + 1;
+  tgid = pid_nr_ns(pid, ns) + 1;
   task = pid_task(pid, PIDTYPE_PID);
   /* What we to know is if the pid we have find is the
    * pid of a thread_group_leader.  Testing for task
@@ -2273,6 +2300,7 @@ int proc_pid_readdir(struct file * filp,
  struct task_struct *reaper = get_proc_task(filp->f_path.dentry->d_inode);
  struct task_struct *task;
  int tgid;
+ struct pid_namespace *ns;

  if (!reaper)
   goto out_no_task;
@@ -2283,11 +2311,12 @@ int proc_pid_readdir(struct file * filp,
    goto out;
  }

+ ns = (struct pid_namespace *)filp->f_dentry->d_sb->s_fs_info;
  tgid = filp->f_pos - TGID_OFFSET;
- for (task = next_tgid(tgid);
+ for (task = next_tgid(tgid, ns);
       task;
-      put_task_struct(task), task = next_tgid(tgid + 1)) {
-  tgid = task->pid;
+      put_task_struct(task), task = next_tgid(tgid + 1, ns)) {
+  tgid = task_pid_nr_ns(task, ns);
   filp->f_pos = tgid + TGID_OFFSET;
```

```
   if (proc_pid_fill_cache(filp, dirent, filldir, task, tgid) < 0) {
    put_task_struct(task);
@@ -2414,6 +2443,7 @@ static struct dentry *proc_task_lookup(s
  struct task_struct *task;
  struct task_struct *leader = get_proc_task(dir);
  unsigned tid;
+ struct pid_namespace *ns;

  if (!leader)
   goto out_no_task;
@@ -2422,8 +2452,9 @@ static struct dentry *proc_task_lookup(s
  if (tid == ~0U)
   goto out;

+ ns = (struct pid_namespace *)dentry->d_sb->s_fs_info;
  rcu_read_lock();
- task = find_task_by_pid(tid);
+ task = find_task_by_pid_ns(tid, ns);
  if (task)
   get_task_struct(task);
  rcu_read_unlock();
@@ -2454,14 +2485,14 @@ out_no_task:
  * threads past it.
  */
 static struct task_struct *first_tid(struct task_struct *leader,
-     int tid, int nr)
+  int tid, int nr, struct pid_namespace *ns)
 {
  struct task_struct *pos;

  rcu_read_lock();
  /* Attempt to start with the pid of a thread */
  if (tid && (nr > 0)) {
-  pos = find_task_by_pid(tid);
+  pos = find_task_by_pid_ns(tid, ns);
   if (pos && (pos->group_leader == leader))
    goto found;
  }
@@ -2530,6 +2561,7 @@ static int proc_task_readdir(struct file
  ino_t ino;
  int tid;
  unsigned long pos = filp->f_pos;  /* avoiding "long long" filp->f_pos */
+ struct pid_namespace *ns;

  task = get_proc_task(inode);
  if (!task)
@@ -2563,12 +2595,13 @@ static int proc_task_readdir(struct file
  /* f_version caches the tgid value that the last readdir call couldn't
```

```
   * return. lseek aka telldir automagically resets f_version to 0.
   */
+ ns = (struct pid_namespace *)filp->f_dentry->d_sb->s_fs_info;
  tid = filp->f_version;
  filp->f_version = 0;
- for (task = first_tid(leader, tid, pos - 2);
+ for (task = first_tid(leader, tid, pos - 2, ns);
       task;
       task = next_tid(task), pos++) {
-  tid = task->pid;
+  tid = task_pid_nr_ns(task, ns);
   if (proc_task_fill_cache(filp, dirent, filldir, task, tid) < 0) {
    /* returning this tgid failed, save it as the first
     * pid for the next readir call */
diff -upr linux-2.6.23-rc1-mm1.orig/include/net/scm.h linux-2.6.23-rc1-mm1-7/include/net/scm.h
--- linux-2.6.23-rc1-mm1.orig/include/net/scm.h 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/include/net/scm.h 2007-07-26 16:36:36.000000000 +0400
@@ -4,6 +4,8 @@
 #include <linux/limits.h>
 #include <linux/net.h>
 #include <linux/security.h>
+#include <linux/pid.h>
+#include <linux/nsproxy.h>

 /* Well, we should have at least one descriptor open
  * to accept passed FDs 8)
@@ -54,7 +56,7 @@ static __inline__ int scm_send(struct so
  struct task_struct *p = current;
  scm->creds.uid = p->uid;
  scm->creds.gid = p->gid;
- scm->creds.pid = p->tgid;
+ scm->creds.pid = task_tgid_vnr(p);
  scm->fp = NULL;
  scm->seq = 0;
  unix_get_peersec_dgram(sock, scm);
diff -upr linux-2.6.23-rc1-mm1.orig/ipc/mqueue.c linux-2.6.23-rc1-mm1-7/ipc/mqueue.c
--- linux-2.6.23-rc1-mm1.orig/ipc/mqueue.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/ipc/mqueue.c 2007-07-26 16:36:36.000000000 +0400
@@ -29,6 +29,8 @@
 #include <linux/audit.h>
 #include <linux/signal.h>
 #include <linux/mutex.h>
+#include <linux/nsproxy.h>
+#include <linux/pid.h>

 #include <net/sock.h>
 #include "util.h"
@@ -336,7 +338,8 @@ static ssize_t mqueue_read_file(struct f
```

```
    (info->notify_owner &&
     info->notify.sigev_notify == SIGEV_SIGNAL) ?
     info->notify.sigev_signo : 0,
-   pid_nr(info->notify_owner));
+   pid_nr_ns(info->notify_owner,
+    current->nsproxy->pid_ns));
  spin_unlock(&info->lock);
  buffer[sizeof(buffer)-1] = '\0';
  slen = strlen(buffer)+1;
@@ -513,7 +516,7 @@ static void __do_notify(struct mqueue_in
    sig_i.si_errno = 0;
    sig_i.si_code = SI_MESGQ;
    sig_i.si_value = info->notify.sigev_value;
-   sig_i.si_pid = current->tgid;
+   sig_i.si_pid = task_pid_vnr(current);
    sig_i.si_uid = current->uid;

    kill_pid_info(info->notify.sigev_signo,
diff -upr linux-2.6.23-rc1-mm1.orig/ipc/msg.c linux-2.6.23-rc1-mm1-7/ipc/msg.c
--- linux-2.6.23-rc1-mm1.orig/ipc/msg.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/ipc/msg.c 2007-07-26 16:36:36.000000000 +0400
@@ -611,7 +611,7 @@ static inline int pipelined_send(struct
     msr->r_msg = ERR_PTR(-E2BIG);
    } else {
     msr->r_msg = NULL;
-    msq->q_lrpid = msr->r_tsk->pid;
+    msq->q_lrpid = task_pid_vnr(msr->r_tsk);
     msq->q_rtime = get_seconds();
     wake_up_process(msr->r_tsk);
     smp_mb();
@@ -695,7 +695,7 @@ long do_msgsnd(int msqid, long mtype, vo
   }
  }

- msq->q_lspid = current->tgid;
+ msq->q_lspid = task_tgid_vnr(current);
  msq->q_stime = get_seconds();

  if (!pipelined_send(msq, msg)) {
@@ -810,7 +810,7 @@ long do_msgrcv(int msqid, long *pmtype,
    list_del(&msg->m_list);
    msq->q_qnum--;
    msq->q_rtime = get_seconds();
-   msq->q_lrpid = current->tgid;
+   msq->q_lrpid = task_tgid_vnr(current);
    msq->q_cbytes -= msg->m_ts;
    atomic_sub(msg->m_ts, &msg_bytes);
    atomic_dec(&msg_hdrs);
```

```
diff -upr linux-2.6.23-rc1-mm1.orig/ipc/sem.c linux-2.6.23-rc1-mm1-7/ipc/sem.c
--- linux-2.6.23-rc1-mm1.orig/ipc/sem.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/ipc/sem.c 2007-07-26 16:36:36.000000000 +0400
@@ -795,7 +795,7 @@ static int semctl_main(struct ipc_namesp
   for (un = sma->undo; un; un = un->id_next)
    un->semadj[semnum] = 0;
   curr->semval = val;
-  curr->sempid = current->tgid;
+  curr->sempid = task_tgid_vnr(current);
   sma->sem_ctime = get_seconds();
   /* maybe some queued-up processes were waiting for this */
   update_queue(sma);
@@ -1196,7 +1196,7 @@ retry_undos:
  if (error)
   goto out_unlock_free;

- error = try_atomic_semop (sma, sops, nsops, un, current->tgid);
+ error = try_atomic_semop (sma, sops, nsops, un, task_tgid_vnr(current));
  if (error <= 0) {
   if (alter && error == 0)
    update_queue (sma);
@@ -1211,7 +1211,7 @@ retry_undos:
  queue.sops = sops;
  queue.nsops = nsops;
  queue.undo = un;
- queue.pid = current->tgid;
+ queue.pid = task_tgid_vnr(current);
  queue.id = semid;
  queue.alter = alter;
  if (alter)
@@ -1382,7 +1382,7 @@ found:
     if (semaphore->semval > SEMVMX)
      semaphore->semval = SEMVMX;
-    semaphore->sempid = current->tgid;
+    semaphore->sempid = task_tgid_vnr(current);
    }
   }
   sma->sem_otime = get_seconds();
diff -upr linux-2.6.23-rc1-mm1.orig/ipc/shm.c linux-2.6.23-rc1-mm1-7/ipc/shm.c
--- linux-2.6.23-rc1-mm1.orig/ipc/shm.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/ipc/shm.c 2007-07-26 16:36:36.000000000 +0400
@@ -168,7 +168,7 @@ static void shm_open(struct vm_area_stru
  shp = shm_lock(sfd->ns, sfd->id);
  BUG_ON(!shp);
  shp->shm_atim = get_seconds();
- shp->shm_lprid = current->tgid;
+ shp->shm_lprid = task_tgid_vnr(current);
```

```
  shp->shm_nattch++;
  shm_unlock(shp);
 }
@@ -213,7 +213,7 @@ static void shm_close(struct vm_area_str
  /* remove from the list of attaches of the shm segment */
  shp = shm_lock(ns, sfd->id);
  BUG_ON(!shp);
- shp->shm_lprid = current->tgid;
+ shp->shm_lprid = task_tgid_vnr(current);
  shp->shm_dtim = get_seconds();
  shp->shm_nattch--;
  if(shp->shm_nattch == 0 &&
@@ -389,7 +389,7 @@ static int newseg (struct ipc_namespace
  if(id == -1)
   goto no_id;

- shp->shm_cprid = current->tgid;
+ shp->shm_cprid = task_tgid_vnr(current);
  shp->shm_lprid = 0;
  shp->shm_atim = shp->shm_dtim = 0;
  shp->shm_ctim = get_seconds();
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/capability.c linux-2.6.23-rc1-mm1-7/kernel/capability.c
--- linux-2.6.23-rc1-mm1.orig/kernel/capability.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/capability.c 2007-07-26 16:36:36.000000000 +0400
@@ -68,8 +68,9 @@ asmlinkage long sys_capget(cap_user_head
  spin_lock(&task_capability_lock);
  read_lock(&tasklist_lock);

- if (pid && pid != current->pid) {
-  target = find_task_by_pid(pid);
+ if (pid && pid != task_pid_vnr(current)) {
+  target = find_task_by_pid_ns(pid,
+    current->nsproxy->pid_ns);
   if (!target) {
    ret = -ESRCH;
    goto out;
@@ -102,7 +103,7 @@ static inline int cap_set_pg(int pgrp_nr
  int found = 0;
  struct pid *pgrp;

- pgrp = find_pid(pgrp_nr);
+ pgrp = find_pid_ns(pgrp_nr, current->nsproxy->pid_ns);
  do_each_pid_task(pgrp, PIDTYPE_PGID, g) {
   target = g;
   while_each_thread(g, target) {
@@ -191,7 +192,7 @@ asmlinkage long sys_capset(cap_user_head
  if (get_user(pid, &header->pid))
   return -EFAULT;
```

```
- if (pid && pid != current->pid && !capable(CAP_SETPCAP))
+ if (pid && pid != task_pid_vnr(current) && !capable(CAP_SETPCAP))
   return -EPERM;

  if (copy_from_user(&effective, &data->effective, sizeof(effective)) ||
@@ -202,8 +203,9 @@ asmlinkage long sys_capset(cap_user_head
  spin_lock(&task_capability_lock);
  read_lock(&tasklist_lock);

- if (pid > 0 && pid != current->pid) {
-  target = find_task_by_pid(pid);
+ if (pid > 0 && pid != task_pid_vnr(current)) {
+  target = find_task_by_pid_ns(pid,
+    current->nsproxy->pid_ns);
   if (!target) {
    ret = -ESRCH;
    goto out;
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c linux-2.6.23-rc1-mm1-7/kernel/exit.c
--- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26 16:36:37.000000000 +0400
@@ -1100,15 +1121,17 @@ asmlinkage void sys_exit_group(int error
 static int eligible_child(pid_t pid, int options, struct task_struct *p)
 {
  int err;
+ struct pid_namespace *ns;

+ ns = current->nsproxy->pid_ns;
  if (pid > 0) {
-  if (p->pid != pid)
+  if (task_pid_nr_ns(p, ns) != pid)
    return 0;
  } else if (!pid) {
-  if (task_pgrp_nr(p) != task_pgrp_nr(current))
+  if (task_pgrp_nr_ns(p, ns) != task_pgrp_vnr(current))
    return 0;
  } else if (pid != -1) {
-  if (task_pgrp_nr(p) != -pid)
+  if (task_pgrp_nr_ns(p, ns) != -pid)
    return 0;
  }

@@ -1179,9 +1202,12 @@ static int wait_task_zombie(struct task_
  unsigned long state;
  int retval;
  int status;
+ struct pid_namespace *ns;
+
```

```
+ ns = current->nsproxy->pid_ns;

  if (unlikely(noreap)) {
-  pid_t pid = p->pid;
+  pid_t pid = task_pid_nr_ns(p, ns);
   uid_t uid = p->uid;
   int exit_code = p->exit_code;
   int why, status;
@@ -1299,7 +1325,7 @@ static int wait_task_zombie(struct task_
    retval = put_user(status, &infop->si_status);
  }
  if (!retval && infop)
-  retval = put_user(p->pid, &infop->si_pid);
+  retval = put_user(task_pid_nr_ns(p, ns), &infop->si_pid);
  if (!retval && infop)
   retval = put_user(p->uid, &infop->si_uid);
  if (retval) {
@@ -1307,7 +1333,7 @@ static int wait_task_zombie(struct task_
   p->exit_state = EXIT_ZOMBIE;
   return retval;
  }
- retval = p->pid;
+ retval = task_pid_nr_ns(p, ns);
  if (p->real_parent != p->parent) {
   write_lock_irq(&tasklist_lock);
   /* Double-check with lock held.  */
@@ -1345,6 +1371,7 @@ static int wait_task_stopped(struct task
        int __user *stat_addr, struct rusage __user *ru)
 {
  int retval, exit_code;
+ struct pid_namespace *ns;

  if (!p->exit_code)
   return 0;
@@ -1363,11 +1390,12 @@ static int wait_task_stopped(struct task
   * keep holding onto the tasklist_lock while we call getrusage and
   * possibly take page faults for user memory.
   */
+ ns = current->nsproxy->pid_ns;
  get_task_struct(p);
  read_unlock(&tasklist_lock);

  if (unlikely(noreap)) {
-  pid_t pid = p->pid;
+  pid_t pid = task_pid_nr_ns(p, ns);
   uid_t uid = p->uid;
   int why = (p->ptrace & PT_PTRACED) ? CLD_TRAPPED : CLD_STOPPED;
```

```
@@ -1438,11 +1466,11 @@ bail_ref:
  if (!retval && infop)
   retval = put_user(exit_code, &infop->si_status);
  if (!retval && infop)
-  retval = put_user(p->pid, &infop->si_pid);
+  retval = put_user(task_pid_nr_ns(p, ns), &infop->si_pid);
  if (!retval && infop)
   retval = put_user(p->uid, &infop->si_uid);
  if (!retval)
-  retval = p->pid;
+  retval = task_pid_nr_ns(p, ns);
  put_task_struct(p);

  BUG_ON(!retval);
@@ -1462,6 +1490,7 @@ static int wait_task_continued(struct ta
  int retval;
  pid_t pid;
  uid_t uid;
+ struct pid_namespace *ns;

  if (unlikely(!p->signal))
   return 0;
@@ -1479,7 +1508,8 @@ static int wait_task_continued(struct ta
   p->signal->flags &= ~SIGNAL_STOP_CONTINUED;
  spin_unlock_irq(&p->sighand->siglock);

- pid = p->pid;
+ ns = current->nsproxy->pid_ns;
+ pid = task_pid_nr_ns(p, ns);
  uid = p->uid;
  get_task_struct(p);
  read_unlock(&tasklist_lock);
@@ -1490,7 +1520,7 @@ static int wait_task_continued(struct ta
   if (!retval && stat_addr)
    retval = put_user(0xffff, stat_addr);
   if (!retval)
-   retval = p->pid;
+   retval = task_pid_nr_ns(p, ns);
  } else {
   retval = wait_noreap_copyout(p, pid, uid,
          CLD_CONTINUED, SIGCONT,
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/fork.c linux-2.6.23-rc1-mm1-7/kernel/fork.c
--- linux-2.6.23-rc1-mm1.orig/kernel/fork.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/fork.c 2007-07-26 16:36:37.000000000 +0400
@@ -1288,7 +1304,7 @@ static struct task_struct *copy_process(
   * TID.  It's too late to back out if this fails.
   */
  if (clone_flags & CLONE_PARENT_SETTID)
```

```
-  put_user(p->pid, parent_tidptr);
+  put_user(task_pid_vnr(p), parent_tidptr);

  proc_fork_connector(p);
  container_post_fork(p);
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/futex.c linux-2.6.23-rc1-mm1-7/kernel/futex.c
--- linux-2.6.23-rc1-mm1.orig/kernel/futex.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/futex.c 2007-07-26 16:36:36.000000000 +0400
@@ -52,6 +52,8 @@
 #include <linux/syscalls.h>
 #include <linux/signal.h>
 #include <linux/module.h>
+#include <linux/pid.h>
+#include <linux/nsproxy.h>
 #include <asm/futex.h>

 #include "rtmutex_common.h"
@@ -652,7 +654,7 @@ static int wake_futex_pi(u32 __user *uad
  if (!(uval & FUTEX_OWNER_DIED)) {
   int ret = 0;

-  newval = FUTEX_WAITERS | new_owner->pid;
+  newval = FUTEX_WAITERS | task_pid_vnr(new_owner);

   curval = cmpxchg_futex_value_locked(uaddr, uval, newval);

@@ -1105,7 +1107,7 @@ static void unqueue_me_pi(struct futex_q
 static int fixup_pi_state_owner(u32 __user *uaddr, struct futex_q *q,
     struct task_struct *curr)
 {
- u32 newtid = curr->pid | FUTEX_WAITERS;
+ u32 newtid = task_pid_vnr(curr) | FUTEX_WAITERS;
  struct futex_pi_state *pi_state = q->pi_state;
  u32 uval, curval, newval;
  int ret;
@@ -1367,7 +1369,7 @@ static int futex_lock_pi(u32 __user *uad
   * (by doing a 0 -> TID atomic cmpxchg), while holding all
   * the locks. It will most likely not succeed.
   */
- newval = current->pid;
+ newval = task_pid_vnr(current);

  curval = cmpxchg_futex_value_locked(uaddr, 0, newval);

@@ -1378,7 +1380,7 @@ static int futex_lock_pi(u32 __user *uad
   * Detect deadlocks. In case of REQUEUE_PI this is a valid
   * situation and we return success to user space.
   */
```

```
- if (unlikely((curval & FUTEX_TID_MASK) == current->pid)) {
+ if (unlikely((curval & FUTEX_TID_MASK) == task_pid_vnr(current))) {
   ret = -EDEADLK;
   goto out_unlock_release_sem;
  }
@@ -1407,7 +1409,7 @@ static int futex_lock_pi(u32 __user *uad
   */
  if (unlikely(ownerdied || !(curval & FUTEX_TID_MASK))) {
   /* Keep the OWNER_DIED bit */
-  newval = (curval & ~FUTEX_TID_MASK) | current->pid;
+  newval = (curval & ~FUTEX_TID_MASK) | task_pid_vnr(current);
   ownerdied = 0;
   lock_taken = 1;
  }
@@ -1586,7 +1588,7 @@ retry:
  /*
   * We release only a lock we actually own:
   */
- if ((uval & FUTEX_TID_MASK) != current->pid)
+ if ((uval & FUTEX_TID_MASK) != task_pid_vnr(current))
   return -EPERM;
  /*
   * First take all the futex related locks:
@@ -1607,7 +1609,7 @@ retry_unlocked:
   * anyone else up:
   */
  if (!(uval & FUTEX_OWNER_DIED))
-  uval = cmpxchg_futex_value_locked(uaddr, current->pid, 0);
+  uval = cmpxchg_futex_value_locked(uaddr, task_pid_vnr(current), 0);


  if (unlikely(uval == -EFAULT))
@@ -1616,7 +1618,7 @@ retry_unlocked:
   * Rare case: we managed to release the lock atomically,
   * no need to wake anyone else up:
   */
- if (unlikely(uval == current->pid))
+ if (unlikely(uval == task_pid_vnr(current)))
   goto out_unlock;

  /*
@@ -1885,7 +1887,7 @@ retry:
  if (get_user(uval, uaddr))
   return -1;

- if ((uval & FUTEX_TID_MASK) == curr->pid) {
+ if ((uval & FUTEX_TID_MASK) == task_pid_vnr(curr)) {
   /*
```

```
    * Ok, this dying thread is truly holding a futex
    * of interest. Set the OWNER_DIED bit atomically
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/ptrace.c linux-2.6.23-rc1-mm1-7/kernel/ptrace.c
--- linux-2.6.23-rc1-mm1.orig/kernel/ptrace.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/ptrace.c 2007-07-26 16:36:36.000000000 +0400
@@ -19,6 +19,7 @@
 #include <linux/security.h>
 #include <linux/signal.h>
 #include <linux/audit.h>
+#include <linux/pid_namespace.h>

 #include <asm/pgtable.h>
 #include <asm/uaccess.h>
@@ -442,7 +443,8 @@ struct task_struct *ptrace_get_task_stru
   return ERR_PTR(-EPERM);

  read_lock(&tasklist_lock);
- child = find_task_by_pid(pid);
+ child = find_task_by_pid_ns(pid,
+    current->nsproxy->pid_ns);
  if (child)
   get_task_struct(child);

diff -upr linux-2.6.23-rc1-mm1.orig/kernel/sched.c linux-2.6.23-rc1-mm1-7/kernel/sched.c
--- linux-2.6.23-rc1-mm1.orig/kernel/sched.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/sched.c 2007-07-26 16:36:36.000000000 +0400
@@ -63,6 +63,7 @@
 #include <linux/delayacct.h>
 #include <linux/reciprocal_div.h>
 #include <linux/unistd.h>
+#include <linux/pid_namespace.h>

 #include <asm/tlb.h>

@@ -1758,7 +1759,7 @@ asmlinkage void schedule_tail(struct tas
  preempt_enable();
 #endif
  if (current->set_child_tid)
-  put_user(current->pid, current->set_child_tid);
+  put_user(task_pid_vnr(current), current->set_child_tid);
 }

 /*
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/signal.c linux-2.6.23-rc1-mm1-7/kernel/signal.c
--- linux-2.6.23-rc1-mm1.orig/kernel/signal.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/signal.c 2007-07-26 16:36:37.000000000 +0400
@@ -696,7 +767,7 @@ static int send_signal(int sig, struct s
    q->info.si_signo = sig;
```

```
    q->info.si_errno = 0;
    q->info.si_code = SI_USER;
-   q->info.si_pid = current->pid;
+   q->info.si_pid = task_pid_vnr(current);
    q->info.si_uid = current->uid;
    break;
   case (unsigned long) SEND_SIG_PRIV:
@@ -1097,7 +1169,7 @@ kill_proc_info(int sig, struct siginfo *
 {
  int error;
  rcu_read_lock();
- error = kill_pid_info(sig, info, find_pid(pid));
+ error = kill_pid_info(sig, info, find_vpid(pid));
  rcu_read_unlock();
  return error;
 }
@@ -1168,9 +1249,9 @@ static int kill_something_info(int sig,
   read_unlock(&tasklist_lock);
   ret = count ? retval : -ESRCH;
  } else if (pid < 0) {
-  ret = kill_pgrp_info(sig, info, find_pid(-pid));
+  ret = kill_pgrp_info(sig, info, find_vpid(-pid));
  } else {
-  ret = kill_pid_info(sig, info, find_pid(pid));
+  ret = kill_pid_info(sig, info, find_vpid(pid));
  }
  rcu_read_unlock();
  return ret;
@@ -1274,7 +1355,12 @@ EXPORT_SYMBOL(kill_pid);
 int
 kill_proc(pid_t pid, int sig, int priv)
 {
- return kill_proc_info(sig, __si_special(priv), pid);
+ int ret;
+
+ rcu_read_lock();
+ ret = kill_pid_info(sig, __si_special(priv), find_pid(pid));
+ rcu_read_unlock();
+ return ret;
 }

 /*
@@ -1452,7 +1538,11 @@ void do_notify_parent(struct task_struct

  info.si_signo = sig;
  info.si_errno = 0;
- info.si_pid = tsk->pid;
+ /*
```

```
+  * we are under tasklist_lock here so our parent is tied to
+  * us and cannot exit and release its namespace.
+  */
+ info.si_pid = task_pid_nr_ns(tsk, tsk->parent->nsproxy->pid_ns);
  info.si_uid = tsk->uid;

  /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1517,7 +1607,11 @@ static void do_notify_parent_cldstop(str

  info.si_signo = SIGCHLD;
  info.si_errno = 0;
- info.si_pid = tsk->pid;
+ /*
+  * we are under tasklist_lock here so our parent is tied to
+  * us and cannot exit and release its namespace.
+  */
+ info.si_pid = task_pid_nr_ns(tsk, tsk->parent->nsproxy->pid_ns);
  info.si_uid = tsk->uid;

  /* FIXME: find out whether or not this is supposed to be c*time. */
@@ -1647,7 +1741,7 @@ void ptrace_notify(int exit_code)
  memset(&info, 0, sizeof info);
  info.si_signo = SIGTRAP;
  info.si_code = exit_code;
- info.si_pid = current->pid;
+ info.si_pid = task_pid_vnr(current);
  info.si_uid = current->uid;

  /* Let the debugger run.  */
@@ -1817,7 +1915,7 @@ relock:
     info->si_signo = signr;
     info->si_errno = 0;
     info->si_code = SI_USER;
-    info->si_pid = current->parent->pid;
+    info->si_pid = task_pid_vnr(current->parent);
     info->si_uid = current->parent->uid;
    }

@@ -2206,7 +2304,7 @@ sys_kill(int pid, int sig)
  info.si_signo = sig;
  info.si_errno = 0;
  info.si_code = SI_USER;
- info.si_pid = current->tgid;
+ info.si_pid = task_tgid_vnr(current);
  info.si_uid = current->uid;

  return kill_something_info(sig, &info, pid);
@@ -2222,12 +2320,12 @@ static int do_tkill(int tgid, int pid, i
```

```
  info.si_signo = sig;
  info.si_errno = 0;
  info.si_code = SI_TKILL;
- info.si_pid = current->tgid;
+ info.si_pid = task_tgid_vnr(current);
  info.si_uid = current->uid;

  read_lock(&tasklist_lock);
- p = find_task_by_pid(pid);
- if (p && (tgid <= 0 || p->tgid == tgid)) {
+ p = find_task_by_pid_ns(pid, current->nsproxy->pid_ns);
+ if (p && (tgid <= 0 || task_tgid_vnr(p) == tgid)) {
   error = check_kill_permission(sig, &info, p);
   /*
    * The null signal is a permissions and process existence
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/sys.c linux-2.6.23-rc1-mm1-7/kernel/sys.c
--- linux-2.6.23-rc1-mm1.orig/kernel/sys.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/sys.c 2007-07-26 16:36:36.000000000 +0400
@@ -152,7 +152,8 @@ asmlinkage long sys_setpriority(int whic
  switch (which) {
   case PRIO_PROCESS:
    if (who)
-    p = find_task_by_pid(who);
+    p = find_task_by_pid_ns(who,
+       current->nsproxy->pid_ns);
    else
     p = current;
    if (p)
@@ -160,7 +161,7 @@ asmlinkage long sys_setpriority(int whic
    break;
   case PRIO_PGRP:
    if (who)
-    pgrp = find_pid(who);
+    pgrp = find_vpid(who);
    else
     pgrp = task_pgrp(current);
    do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
@@ -209,7 +210,8 @@ asmlinkage long sys_getpriority(int whic
  switch (which) {
   case PRIO_PROCESS:
    if (who)
-    p = find_task_by_pid(who);
+    p = find_task_by_pid_ns(who,
+       current->nsproxy->pid_ns);
    else
     p = current;
    if (p) {
@@ -220,7 +222,7 @@ asmlinkage long sys_getpriority(int whic
```

```
    break;
   case PRIO_PGRP:
    if (who)
-    pgrp = find_pid(who);
+    pgrp = find_vpid(who);
    else
     pgrp = task_pgrp(current);
    do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
@@ -916,7 +918,7 @@ asmlinkage long sys_setpgid(pid_t pid, p
  int err = -EINVAL;

  if (!pid)
-  pid = group_leader->pid;
+  pid = task_pid_vnr(group_leader);
  if (!pgid)
   pgid = pid;
  if (pgid < 0)
@@ -928,7 +930,7 @@ asmlinkage long sys_setpgid(pid_t pid, p
  write_lock_irq(&tasklist_lock);

  err = -ESRCH;
- p = find_task_by_pid(pid);
+ p = find_task_by_pid_ns(pid, current->nsproxy->pid_ns);
  if (!p)
   goto out;

@@ -955,7 +957,8 @@ asmlinkage long sys_setpgid(pid_t pid, p

  if (pgid != pid) {
   struct task_struct *g =
-   find_task_by_pid_type(PIDTYPE_PGID, pgid);
+   find_task_by_pid_type_ns(PIDTYPE_PGID, pgid,
+     current->nsproxy->pid_ns);

   if (!g || task_session(g) != task_session(group_leader))
    goto out;
@@ -966,9 +969,12 @@ asmlinkage long sys_setpgid(pid_t pid, p
   goto out;

  if (task_pgrp_nr(p) != pgid) {
+  struct pid *pid;
+
   detach_pid(p, PIDTYPE_PGID);
-  p->signal->pgrp = pgid;
-  attach_pid(p, PIDTYPE_PGID, find_pid(pgid));
+  pid = find_vpid(pgid);
+  attach_pid(p, PIDTYPE_PGID, pid);
+  p->signal->pgrp = pid_nr(pid);
```

```
  }

  err = 0;
@@ -981,19 +987,20 @@ out:
 asmlinkage long sys_getpgid(pid_t pid)
 {
  if (!pid)
-  return task_pgrp_nr(current);
+  return task_pgrp_vnr(current);
  else {
   int retval;
   struct task_struct *p;

   read_lock(&tasklist_lock);
-  p = find_task_by_pid(pid);
+  p = find_task_by_pid_ns(pid,
+    current->nsproxy->pid_ns);

   retval = -ESRCH;
   if (p) {
    retval = security_task_getpgid(p);
    if (!retval)
-    retval = task_pgrp_nr(p);
+    retval = task_pgrp_vnr(p);
   }
   read_unlock(&tasklist_lock);
   return retval;
@@ -1005,7 +1012,7 @@ asmlinkage long sys_getpgid(pid_t pid)
 asmlinkage long sys_getpgrp(void)
 {
  /* SMP - assuming writes are word atomic this is fine */
- return task_pgrp_nr(current);
+ return task_pgrp_vnr(current);
 }

 #endif
@@ -1013,19 +1020,20 @@ asmlinkage long sys_getpgrp(void)
 asmlinkage long sys_getsid(pid_t pid)
 {
  if (!pid)
-  return task_session_nr(current);
+  return task_session_vnr(current);
  else {
   int retval;
   struct task_struct *p;

   read_lock(&tasklist_lock);
-  p = find_task_by_pid(pid);
```

```
+  p = find_task_by_pid_ns(pid,
+    current->nsproxy->pid_ns);

   retval = -ESRCH;
   if (p) {
    retval = security_task_getsid(p);
    if (!retval)
-    retval = task_session_nr(p);
+    retval = task_session_vnr(p);
   }
   read_unlock(&tasklist_lock);
   return retval;
@@ -1062,7 +1070,7 @@ asmlinkage long sys_setsid(void)
  group_leader->signal->tty = NULL;
  spin_unlock(&group_leader->sighand->siglock);

- err = task_pgrp_nr(group_leader);
+ err = task_pgrp_vnr(group_leader);
 out:
  write_unlock_irq(&tasklist_lock);
  return err;
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/sysctl.c linux-2.6.23-rc1-mm1-7/kernel/sysctl.c
--- linux-2.6.23-rc1-mm1.orig/kernel/sysctl.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/sysctl.c 2007-07-26 16:36:36.000000000 +0400
@@ -2331,7 +2331,7 @@ static int proc_do_cad_pid(ctl_table *ta
  pid_t tmp;
  int r;

- tmp = pid_nr(cad_pid);
+ tmp = pid_nr_ns(cad_pid, current->nsproxy->pid_ns);

  r = __do_proc_dointvec(&tmp, table, write, filp, buffer,
           lenp, ppos, NULL, NULL);
diff -upr linux-2.6.23-rc1-mm1.orig/kernel/timer.c linux-2.6.23-rc1-mm1-7/kernel/timer.c
--- linux-2.6.23-rc1-mm1.orig/kernel/timer.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/kernel/timer.c 2007-07-26 16:36:36.000000000 +0400
@@ -37,6 +37,7 @@
 #include <linux/tick.h>
 #include <linux/kallsyms.h>
 #include <linux/kgdb.h>
+#include <linux/pid_namespace.h>

 #include <asm/uaccess.h>
 #include <asm/unistd.h>
@@ -957,7 +958,7 @@ asmlinkage unsigned long sys_alarm(unsig
  */
 asmlinkage long sys_getpid(void)
 {
```

```
- return current->tgid;
+ return task_tgid_vnr(current);
 }

 /*
@@ -971,7 +972,7 @@ asmlinkage long sys_getppid(void)
  int pid;

  rcu_read_lock();
- pid = rcu_dereference(current->real_parent)->tgid;
+ pid = task_ppid_nr_ns(current, current->nsproxy->pid_ns);
  rcu_read_unlock();

  return pid;
@@ -1103,7 +1104,7 @@ EXPORT_SYMBOL(schedule_timeout_uninterru
 /* Thread ID - the internal kernel "pid" */
 asmlinkage long sys_gettid(void)
 {
- return current->pid;
+ return task_pid_vnr(current);
 }

 /**
diff -upr linux-2.6.23-rc1-mm1.orig/net/core/scm.c linux-2.6.23-rc1-mm1-7/net/core/scm.c
--- linux-2.6.23-rc1-mm1.orig/net/core/scm.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/net/core/scm.c 2007-07-26 16:36:36.000000000 +0400
@@ -24,6 +24,8 @@
 #include <linux/interrupt.h>
 #include <linux/netdevice.h>
 #include <linux/security.h>
+#include <linux/pid.h>
+#include <linux/nsproxy.h>

 #include <asm/system.h>
 #include <asm/uaccess.h>
@@ -42,7 +44,7 @@

 static __inline__ int scm_check_creds(struct ucred *creds)
 {
- if ((creds->pid == current->tgid || capable(CAP_SYS_ADMIN)) &&
+ if ((creds->pid == task_tgid_vnr(current) || capable(CAP_SYS_ADMIN)) &&
     ((creds->uid == current->uid || creds->uid == current->euid ||
       creds->uid == current->suid) || capable(CAP_SETUID)) &&
     ((creds->gid == current->gid || creds->gid == current->egid ||
diff -upr linux-2.6.23-rc1-mm1.orig/net/unix/af_unix.c linux-2.6.23-rc1-mm1-7/net/unix/af_unix.c
--- linux-2.6.23-rc1-mm1.orig/net/unix/af_unix.c 2007-07-26 16:34:45.000000000 +0400
+++ linux-2.6.23-rc1-mm1-7/net/unix/af_unix.c 2007-07-26 16:36:36.000000000 +0400
@@ -456,7 +456,7 @@ static int unix_listen(struct socket *so
```

```
  sk->sk_max_ack_backlog = backlog;
  sk->sk_state  = TCP_LISTEN;
  /* set credentials so connect can copy them */
- sk->sk_peercred.pid = current->tgid;
+ sk->sk_peercred.pid = task_tgid_vnr(current);
  sk->sk_peercred.uid = current->euid;
  sk->sk_peercred.gid = current->egid;
  err = 0;
@@ -1103,7 +1103,7 @@ restart:
  unix_peer(newsk) = sk;
  newsk->sk_state  = TCP_ESTABLISHED;
  newsk->sk_type  = sk->sk_type;
- newsk->sk_peercred.pid = current->tgid;
+ newsk->sk_peercred.pid = task_tgid_vnr(current);
  newsk->sk_peercred.uid = current->euid;
  newsk->sk_peercred.gid = current->egid;
  newu = unix_sk(newsk);
@@ -1164,7 +1164,7 @@ static int unix_socketpair(struct socket
  sock_hold(skb);
  unix_peer(ska)=skb;
  unix_peer(skb)=ska;
- ska->sk_peercred.pid = skb->sk_peercred.pid = current->tgid;
+ ska->sk_peercred.pid = skb->sk_peercred.pid = task_tgid_vnr(current);
  ska->sk_peercred.uid = skb->sk_peercred.uid = current->euid;
  ska->sk_peercred.gid = skb->sk_peercred.gid = current->egid;
```

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Dave Hansen on Thu, 26 Jul 2007 16:10:39 GMT
View Forum Message <> Reply to Message

On Thu, 2007-07-26 at 18:46 +0400, Pavel Emelyanov wrote:
> +	write_unlock_irq(&tasklist_lock);
> +	exit_task_namespaces(tsk);
> +	write_lock_irq(&tasklist_lock);

Are there any other side-effects of doing this? What was the
tasklist_lock protecting here when it was released?

-- Dave

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Oleg Nesterov on Thu, 26 Jul 2007 16:47:24 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:

>
> Make task release its namespaces after it has reparented all his
> children to child_reaper, but before it notifies its parent about
> its death.
>
> The reason to release namespaces after reparenting is that when task
> exits it may send a signal to its parent (SIGCHLD), but if the parent
> has already exited its namespaces there will be no way to decide what
> pid to dever to him - parent can be from different namespace.
>
> The reason to release namespace before notifying the parent it that
> when task sends a SIGCHLD to parent it can call wait() on this taks
> and release it. But releasing the mnt namespace implies dropping
> of all the mounts in the mnt namespace and NFS expects the task to
> have valid sighand pointer.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> exit.c | 5 ++++-
> 1 files changed, 4 insertions(+), 1 deletion(-)
>
> diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c
> linux-2.6.23-rc1-mm1-7/kernel/exit.c
> --- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26
> 16:34:45.000000000 +0400
> +++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26
> 16:36:37.000000000 +0400
> @@ -788,6 +804,10 @@ static void exit_notify(struct task_stru
> BUG_ON(!list_empty(&tsk->children));
> BUG_ON(!list_empty(&tsk->ptrace_children));
>
> + write_unlock_irq(&tasklist_lock);
> + exit_task_namespaces(tsk);
> + write_lock_irq(&tasklist_lock);

No.

We "cleared" our ->children/->ptrace_children lists. Now suppose that
another thread dies, and its forget_original_parent() choose us as a
new reaper before we re-take tasklist.

I'll try to read other patches tomorrow, but I can't avoid a stupid
question: can we have a CONFIG_ for that? This series adds a lot of
complications.

OK, I guess the answer is "it is very difficult t achieve", but can't

help myself :)

Oleg.

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by dev on Thu, 26 Jul 2007 16:59:13 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>
>>Make task release its namespaces after it has reparented all his
>>children to child_reaper, but before it notifies its parent about
>>its death.
>>
>>The reason to release namespaces after reparenting is that when task
>>exits it may send a signal to its parent (SIGCHLD), but if the parent
>>has already exited its namespaces there will be no way to decide what
>>pid to dever to him - parent can be from different namespace.
>>
>>The reason to release namespace before notifying the parent it that
>>when task sends a SIGCHLD to parent it can call wait() on this taks
>>and release it. But releasing the mnt namespace implies dropping
>>of all the mounts in the mnt namespace and NFS expects the task to
>>have valid sighand pointer.
>>
>>Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>>
>>---
>>
>>exit.c | 5 ++++-
>>1 files changed, 4 insertions(+), 1 deletion(-)
>>
>>diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c
>>linux-2.6.23-rc1-mm1-7/kernel/exit.c
>>--- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26
>>16:34:45.000000000 +0400
>>+++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26
>>16:36:37.000000000 +0400
>>@@ -788,6 +804,10 @@ static void exit_notify(struct task_stru
>> BUG_ON(!list_empty(&tsk->children));
>> BUG_ON(!list_empty(&tsk->ptrace_children));
>>
>>+ write_unlock_irq(&tasklist_lock);
>>+ exit_task_namespaces(tsk);
>>+ write_lock_irq(&tasklist_lock);
>
>

>
> No.
>
> We "cleared" our ->children/->ptrace_children lists. Now suppose that
> another thread dies, and its forget_original_parent() choose us as a
> new reaper before we re-take tasklist.
>
> I'll try to read other patches tomorrow, but I can't avoid a stupid
> question: can we have a CONFIG_ for that? This series adds a lot of
> complications.

the was a request from many people including Andrew that CONFIG_XXX
is bad approach.

> OK, I guess the answer is "it is very difficult t achieve", but can't
> help myself :)

First versions of Pavel's patches had CONFIG_XXX for this.

Thanks,
Kirill

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

---

Subject: Re: [PATCH 7/15] Helpers to obtain pid numbers
Posted by Dave Hansen on Thu, 26 Jul 2007 19:03:01 GMT
View Forum Message <> Reply to Message

On Thu, 2007-07-26 at 18:51 +0400, Pavel Emelyanov wrote:
>
> + * pid_nr()   : global id, i.e. the id seen from the init namespace;
> + * pid_vnr()   : virtual id, i.e. the id seen from the namespace this pid
> + *   belongs to. this only makes sence when called in the
> + *   context of the task that belongs to the same namespace;

Can we give these some better names? I think "virtual" is pretty bad,
especially if you consider the multiple level of pid namespaces that we
might have some day. Processes can belong to multiple pid namespaces,
and thus have multiple "virtual" ids.

Even though it will make the names longer, I think we need something in
the names to say that "pid_nr()" is the top-level, global, init_pid_ns
number. "pid_vnr()" is the pid for the lowest pid namespace in the
hierarchy. Suka called this an "active pid namespace" because that is
where the task actively interacts with its peers. But, I'm open to

other suggestions, too.

When writing code, people are going to need to know which one to use:
pid_nr() or pid_vnr(). We can document the functions, but the names
will help much more than any documentation.

-- Dave

Subject: Re: [PATCH 8/15] Helpers to find the task by its numerical ids
Posted by Dave Hansen on Thu, 26 Jul 2007 19:05:41 GMT
View Forum Message <> Reply to Message

On Thu, 2007-07-26 at 18:51 +0400, Pavel Emelyanov wrote:
> +extern struct task_struct *find_task_by_pid_type_ns(int type, int pid,
> + struct pid_namespace *ns);
> +
> +#define find_task_by_pid_ns(nr, ns) \
> + find_task_by_pid_type_ns(PIDTYPE_PID, nr, ns)
> +#define find_task_by_pid_type(type, nr) \
> + find_task_by_pid_type_ns(type, nr, &init_pid_ns)
> +#define find_task_by_pid(nr) \
> + find_task_by_pid_type(PIDTYPE_PID, nr)
> +
> extern void __set_special_pids(pid_t session, pid_t pgrp);

Do these _have_ to be macros?

> /* per-UID process charging. */
> diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
> --- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
> +++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
> @@ -204,19 +221,20 @@ static void delayed_put_pid(struct rcu_h
> goto out;
> }
>
> -struct pid * fastcall find_pid(int nr)
> +struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
> {
> struct hlist_node *elem;
> - struct pid *pid;
> + struct upid *pnr;
> +
> + hlist_for_each_entry_rcu(pnr, elem,
> + &pid_hash[pid_hashfn(nr, ns)], pid_chain)
> + if (pnr->nr == nr && pnr->ns == ns)
> + return container_of(pnr, struct pid,

> +   numbers[ns->level]);

Do we do this loop anywhere else? Should we have a for_each_pid() that
makes this a little less messy?

> - hlist_for_each_entry_rcu(pid, elem,
> -   &pid_hash[pid_hashfn(nr)], pid_chain) {
> - if (pid->nr == nr)
> -   return pid;
> - }
>   return NULL;
> }
> -EXPORT_SYMBOL_GPL(find_pid);
> +EXPORT_SYMBOL_GPL(find_pid_ns);
>
> /*
>   * attach_pid() must be called with the tasklist_lock write-held.
> @@ -318,12 +355,13 @@ struct task_struct * fastcall pid_task(s
> /*
>   * Must be called under rcu_read_lock() or with tasklist_lock read-held.
>   */
> -struct task_struct *find_task_by_pid_type(int type, int nr)
> +struct task_struct *find_task_by_pid_type_ns(int type, int nr,
> + struct pid_namespace *ns)
> {
> - return pid_task(find_pid(nr), type);
> + return pid_task(find_pid_ns(nr, ns), type);
> }
>
> -EXPORT_SYMBOL(find_task_by_pid_type);
> +EXPORT_SYMBOL(find_task_by_pid_type_ns);
>
> struct pid *get_task_pid(struct task_struct *task, enum pid_type type)
> {
> @@ -342,7 +426,7 @@ struct pid *find_get_pid(pid_t nr)
>   struct pid *pid;
>
>   rcu_read_lock();
> - pid = get_pid(find_pid(nr));
> + pid = get_pid(find_vpid(nr));
>   rcu_read_unlock();

OK, I think this is really confusing. If find_get_pid() finds vpids,
should we not call it find_get_vpid()?

-- Dave

Subject: Re: [RFC][PATCH 0/15] Pid namespaces
Posted by Sukadev Bhattiprolu on Fri, 27 Jul 2007 04:22:13 GMT
View Forum Message <> Reply to Message

Pavel,

We seem to have a memory leak. Its new in this patchset (i.e the
following test ran fine on the 2.6.22-rc6-mm1 patchset).

To repro: run "pidns_exec ./mypid" in a tight-loop - where mypid.c
is:

```
#include <stdio.h>
#include <unistd.h>

main()
{
	printf("Pid %d, Ppid %d, Pgid %d, Sid %d\n",
		getpid(), getppid(), getpgid(0), getsid(0));
}
```

I ran into OOM in about 30 mins. I am still investigating.

BTW, can we include a simple test program like the pidns_exec in this
patch-0 for whoever want to play with pidns ?

Suka

Pavel Emelianov [xemul@openvz.org] wrote:
| (Comment is taken from Sukadev's patchset-v3)
|
| A pid namespace is a "view" of a particular set of tasks on the system.
| They work in a similar way to filesystem namespaces. A file (or a process)
| can be accessed in multiple namespaces, but it may have a different name
| in each. In a filesystem, this name might be /etc/passwd in one namespace,
| but /chroot/etc/passwd in another.
|
| For processes, a process may have pid 1234 in one namespace, but be pid 1
| in another. This allows new pid namespaces to have basically arbitrary
| pids, and not have to worry about what pids exist in other namespaces.
| This is essential for checkpoint/restart where a restarted process's pid
| might collide with an existing process on the system's pid.
|
| In this particular implementation, pid namespaces have a parent-child
| relationship, just like processes. A process in a pid namespace may see
| all of the processes in the same namespace, as well as all of the processes
| in all of the namespaces which are children of its namespace. Processes may
| not, however, see others which are in their parent's namespace, but not in
| their own. The same goes for sibling namespaces.

|
| This set is based on my patches, I sent before, but it includes some
| comments
| and patches that I received from Sukadev. Sukadev, please, add your
| Acked-by,
| Signed-off-by or From, to patches you want (everybody is also welcome :)).
|
| The set is based on 2.6.23-rc1-mm1, which already has some preparation
| patches
| for pid namespaces. After the review and fixing all the comments, this set
| will be benchmarked and sent to Andrew for inclusion in -mm tree.
|
| Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
| Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

Subject: Re: [PATCH 15/15] Hooks over the code to show correct values to user
Posted by Sukadev Bhattiprolu on Fri, 27 Jul 2007 05:57:36 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| diff -upr linux-2.6.23-rc1-mm1.orig/fs/proc/array.c
| linux-2.6.23-rc1-mm1-7/fs/proc/array.c
| --- linux-2.6.23-rc1-mm1.orig/fs/proc/array.c 2007-07-26
| 16:34:45.000000000 +0400
| +++ linux-2.6.23-rc1-mm1-7/fs/proc/array.c 2007-07-26
| 16:36:36.000000000 +0400
| @@ -77,6 +77,7 @@
| #include <linux/cpuset.h>
| #include <linux/rcupdate.h>
| #include <linux/delayacct.h>
| +#include <linux/pid_namespace.h>
|
| #include <asm/pgtable.h>
| #include <asm/processor.h>
| @@ -161,7 +162,9 @@ static inline char *task_state(struct ta
| struct group_info *group_info;
| int g;
| struct fdtable *fdt = NULL;
| + struct pid_namespace *ns;
|
| + ns = current->nsproxy->pid_ns;
| rcu_read_lock();
| buffer += sprintf(buffer,
| "State:\t%s\n"
| @@ -172,9 +175,12 @@ static inline char *task_state(struct ta
| "Uid:\t%d\t%d\t%d\t%d\n"
| "Gid:\t%d\t%d\t%d\t%d\n",

```
|   get_task_state(p),
| -  p->tgid, p->pid,
| -  pid_alive(p) ? rcu_dereference(p->real_parent)->tgid : 0,
| -  pid_alive(p) && p->ptrace ? rcu_dereference(p->parent)->pid
| : 0,
| +         task_tgid_nr_ns(p, ns),
| +  task_pid_nr_ns(p, ns),
| +         pid_alive(p) ?
| +   task_ppid_nr_ns(p, ns) : 0,
| +  pid_alive(p) && p->ptrace ?
| +   task_tgid_nr_ns(rcu_dereference(p->parent), ns) : 0,
```

Hmm. I think we have this reversed here. We should print the 'tgid' of task->real_parent and 'pid' of task->parent.

Subject: Re: [RFC][PATCH 0/15] Pid namespaces
Posted by Sukadev Bhattiprolu on Fri, 27 Jul 2007 06:08:56 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com [sukadev@us.ibm.com] wrote:

```
| Pavel,
|
| We seem to have a memory leak. Its new in this patchset (i.e the
| following test ran fine on the 2.6.22-rc6-mm1 patchset).
|
| To repro: run "pidns_exec ./mypid" in a tight-loop  - where mypid.c
| is:
|
| #include <stdio.h>
| #include <unistd.h>
|
| main()
| {
|         printf("Pid %d, Ppid %d, Pgid %d, Sid %d\n",
|                 getpid(), getppid(), getpgid(0), getsid(0));
| }
|
| I ran into OOM in about 30 mins. I am still investigating.
|
| BTW, can we include a simple test program like the pidns_exec in this
| patch-0 for whoever want to play with pidns ?
|
| Suka
|
```

I think the problem is in create_pid_namespace(). kref_init() sets the refcount to 1 and then we do a get_pid_ns() which sets it to 2.

free_nsproxy() frees one of this references, but the other is never
freed.

This patch seems to fix the leak. The patch also creates a slab cache
for the pid_namespace

Create a slab-cache for 'struct pid_namespace' and fix a memory leak
due to an extra reference in create_pid_namespace().

---
 kernel/pid.c | 10 ++++++----
 1 file changed, 6 insertions(+), 4 deletions(-)

Index: lx26-23-rc1-mm1/kernel/pid.c
===
--- lx26-23-rc1-mm1.orig/kernel/pid.c 2007-07-26 20:08:16.000000000 -0700
+++ lx26-23-rc1-mm1/kernel/pid.c 2007-07-26 22:31:42.000000000 -0700
@@ -35,6 +35,7 @@
 static struct hlist_head *pid_hash;
 static int pidhash_shift;
 struct pid init_struct_pid = INIT_STRUCT_PID;
+static struct kmem_cache *pid_ns_cachep;

 int pid_max = PID_MAX_DEFAULT;

@@ -525,7 +526,7 @@ static struct pid_namespace *create_pid_
  struct pid_namespace *ns;
  int i;

- ns = kmalloc(sizeof(struct pid_namespace), GFP_KERNEL);
+ ns = kmem_cache_alloc(pid_ns_cachep, GFP_KERNEL);
  if (ns == NULL)
   goto out;

@@ -544,7 +545,6 @@ static struct pid_namespace *create_pid_

  set_bit(0, ns->pidmap[0].page);
  atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
- get_pid_ns(ns);

  for (i = 1; i < PIDMAP_ENTRIES; i++) {
   ns->pidmap[i].page = 0;
@@ -556,7 +556,7 @@ static struct pid_namespace *create_pid_
 out_free_map:
  kfree(ns->pidmap[0].page);

```
 out_free:
- kfree(ns);
+ kmem_cache_free(pid_ns_cachep, ns);
 out:
  return ERR_PTR(-ENOMEM);
 }
@@ -567,7 +567,7 @@ static void destroy_pid_namespace(struct

  for (i = 0; i < PIDMAP_ENTRIES; i++)
   kfree(ns->pidmap[i].page);
- kfree(ns);
+ kmem_cache_free(pid_ns_cachep, ns);
 }

   struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
@@ -687,4 +687,6 @@ void __init pidmap_init(void)
  init_pid_ns.pid_cachep = create_pid_cachep(1);
  if (init_pid_ns.pid_cachep == NULL)
   panic("Can't create pid_1 cachep\n");
+
+ pid_ns_cachep = KMEM_CACHE(pid_namespace, SLAB_PANIC);
 }
```

Subject: Re: [PATCH 12/15] Miscelaneous stuff for pid namespaces
Posted by Sukadev Bhattiprolu on Fri, 27 Jul 2007 06:22:13 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| This includes
| * the headers dependency fix
| * task_child_reaper() correct declaration
| * fixes for is_global_init() and is_container_init()
|
| Maybe this should come before all the other stuff...
|
| Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
|
| ---
|
| include/linux/pid_namespace.h | 4 ++--
| include/linux/sched.h | 4 ++--
| kernel/pid.c | 16 ++++++++++++---
| 3 files changed, 17 insertions(+), 7 deletions(-)

```
|
| diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h
| linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h
| --- linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h 2007-07-26
| 16:34:45.000000000 +0400
| +++ linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h 2007-07-26
| 16:36:36.000000000 +0400
| @@ -4,7 +4,6 @@
| #include <linux/sched.h>
| #include <linux/mm.h>
| #include <linux/threads.h>
| -#include <linux/pid.h>
| #include <linux/nsproxy.h>
| #include <linux/kref.h>
|
| @@ -46,7 +53,8 @@ static inline struct pid_namespace *task
|
| static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
| {
| - return init_pid_ns.child_reaper;
| + BUG_ON(tsk != current);
| + return tsk->nsproxy->pid_ns->child_reaper;
| }
|
| #endif /* _LINUX_PID_NS_H */
| diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/sched.h
| linux-2.6.23-rc1-mm1-7/include/linux/sched.h
| --- linux-2.6.23-rc1-mm1.orig/include/linux/sched.h 2007-07-26
| 16:34:45.000000000 +0400
| +++ linux-2.6.23-rc1-mm1-7/include/linux/sched.h 2007-07-26
| 16:36:37.000000000 +0400
| @@ -1277,13 +1366,13 @@ static inline int pid_alive(struct task_
|  *
|  * TODO: We should inline this function after some cleanups in
|  pid_namespace.h
|  */
```

We can now remove this TODO.

```
| -extern int is_global_init(struct task_struct *tsk);
| +extern int is_container_init(struct task_struct *tsk);
|
| /*
|  * is_container_init:
|  * check whether in the task is init in its own pid namespace.
|  */
```

The function header describes is_container_init() but the function

is actually is_global_init(). The opp is true for the function
heaer of is_container_init() above.

| -static inline int is_container_init(struct task_struct *tsk)
| +static inline int is_global_init(struct task_struct *tsk)
| {
| return tsk->pid == 1;
| }
| diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c
| linux-2.6.23-rc1-mm1-7/kernel/pid.c
| --- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26
| 16:34:45.000000000 +0400
| +++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26
| 16:36:37.000000000 +0400
| @@ -60,11 +62,21 @@ static inline int mk_pid(struct pid_name
| };
| EXPORT_SYMBOL(init_pid_ns);
|
| -int is_global_init(struct task_struct *tsk)
| +int is_container_init(struct task_struct *tsk)
| {
| - return tsk == init_pid_ns.child_reaper;
| + int ret;

Initialize ret = 0 here would save a line of code below :-)

| + struct pid *pid;
| +
| + ret = 0;
| + rcu_read_lock();
| + pid = task_pid(tsk);
| + if (pid != NULL && pid->numbers[pid->level].nr == 1)
| + ret = 1;
| + rcu_read_unlock();
| +
| + return ret;
| }
| -EXPORT_SYMBOL(is_global_init);
| +EXPORT_SYMBOL(is_container_init);
|
| /*
| * Note: disable interrupts while the pidmap_lock is held as an

---

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by xemul on Fri, 27 Jul 2007 06:38:02 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2007-07-26 at 18:46 +0400, Pavel Emelyanov wrote:
>> +	write_unlock_irq(&tasklist_lock);
>> +	exit_task_namespaces(tsk);
>> +	write_lock_irq(&tasklist_lock);
>
> Are there any other side-effects of doing this? What was the

Looks like there are :(Oleg pointed out, that if any thread dies
it can re-parent all its children to this one, who already released
its namespaces... :(

> tasklist_lock protecting here when it was released?
>
> -- Dave
>
>

Subject: Re: [PATCH 7/15] Helpers to obtain pid numbers
Posted by Pavel Emelianov on Fri, 27 Jul 2007 06:40:47 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2007-07-26 at 18:51 +0400, Pavel Emelyanov wrote:
>> + * pid_nr()   : global id, i.e. the id seen from the init namespace;
>> + * pid_vnr()   : virtual id, i.e. the id seen from the namespace this pid
>> + *   belongs to. this only makes sence when called in the
>> + *   context of the task that belongs to the same namespace;
>
> Can we give these some better names? I think "virtual" is pretty bad,
> especially if you consider the multiple level of pid namespaces that we
> might have some day. Processes can belong to multiple pid namespaces,
> and thus have multiple "virtual" ids.

We do have them now (multiple levels). The "virtual" stands for "the one
that task sees by his own".

> Even though it will make the names longer, I think we need something in
> the names to say that "pid_nr()" is the top-level, global, init_pid_ns
> number. "pid_vnr()" is the pid for the lowest pid namespace in the
> hierarchy.

That's it.

> Suka called this an "active pid namespace" because that is
> where the task actively interacts with its peers. But, I'm open to
> other suggestions, too.

>
> When writing code, people are going to need to know which one to use:
> pid_nr() or pid_vnr(). We can document the functions, but the names
> will help much more than any documentation.

I do not mind, but what other names can we have? pid_nr() for init namespace,
pid_nr_ns for arbitrary namespace and pid_<what> for the lower-level namespace?

>
> -- Dave
>
>

Thanks,
Pavel

Subject: Re: [PATCH 8/15] Helpers to find the task by its numerical ids
Posted by Pavel Emelianov on Fri, 27 Jul 2007 06:43:38 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2007-07-26 at 18:51 +0400, Pavel Emelyanov wrote:
>> +extern struct task_struct *find_task_by_pid_type_ns(int type, int pid,
>> + struct pid_namespace *ns);
>> +
>> +#define find_task_by_pid_ns(nr, ns) \
>> + find_task_by_pid_type_ns(PIDTYPE_PID, nr, ns)
>> +#define find_task_by_pid_type(type, nr) \
>> + find_task_by_pid_type_ns(type, nr, &init_pid_ns)
>> +#define find_task_by_pid(nr) \
>> + find_task_by_pid_type(PIDTYPE_PID, nr)
>> +
>> extern void __set_special_pids(pid_t session, pid_t pgrp);
>
> Do these _have_ to be macros?

No, but why not?
I can make them static inline functions, but why are macros that bad?

>> /* per-UID process charging. */
>> diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c linux-2.6.23-rc1-mm1-7/kernel/pid.c
>> --- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
>> +++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
>> @@ -204,19 +221,20 @@ static void delayed_put_pid(struct rcu_h
>> goto out;
>> }
>>

>> -struct pid * fastcall find_pid(int nr)
>> +struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
>> {
>>   struct hlist_node *elem;
>> - struct pid *pid;
>> + struct upid *pnr;
>> +
>> + hlist_for_each_entry_rcu(pnr, elem,
>> +   &pid_hash[pid_hashfn(nr, ns)], pid_chain)
>> + if (pnr->nr == nr && pnr->ns == ns)
>> +   return container_of(pnr, struct pid,
>> +   numbers[ns->level]);
>
> Do we do this loop anywhere else? Should we have a for_each_pid() that
> makes this a little less messy?

No. Iteration over the hash chain happens here only.

>> - hlist_for_each_entry_rcu(pid, elem,
>> -   &pid_hash[pid_hashfn(nr)], pid_chain) {
>> - if (pid->nr == nr)
>> -   return pid;
>> - }
>>   return NULL;
>> }
>> -EXPORT_SYMBOL_GPL(find_pid);
>> +EXPORT_SYMBOL_GPL(find_pid_ns);
>>
>> /*
>>   * attach_pid() must be called with the tasklist_lock write-held.
>> @@ -318,12 +355,13 @@ struct task_struct * fastcall pid_task(s
>> /*
>>   * Must be called under rcu_read_lock() or with tasklist_lock read-held.
>>   */
>> -struct task_struct *find_task_by_pid_type(int type, int nr)
>> +struct task_struct *find_task_by_pid_type_ns(int type, int nr,
>> + struct pid_namespace *ns)
>> {
>> - return pid_task(find_pid(nr), type);
>> + return pid_task(find_pid_ns(nr, ns), type);
>> }
>>
>> -EXPORT_SYMBOL(find_task_by_pid_type);
>> +EXPORT_SYMBOL(find_task_by_pid_type_ns);
>>
>> struct pid *get_task_pid(struct task_struct *task, enum pid_type type)
>> {
>> @@ -342,7 +426,7 @@ struct pid *find_get_pid(pid_t nr)

>>   struct pid *pid;
>>
>>   rcu_read_lock();
>> - pid = get_pid(find_pid(nr));
>> + pid = get_pid(find_vpid(nr));
>>   rcu_read_unlock();
>
> OK, I think this is really confusing. If find_get_pid() finds vpids,
> should we not call it find_get_vpid()?

I'd better make it find_get_pid_ns() with two arguments and made a couple
of macros (or static inlines) for global search and local search.

> -- Dave
>
>

Thanks,
Pavel

Subject: Re: [PATCH 15/15] Hooks over the code to show correct values to user
Posted by Pavel Emelianov on Fri, 27 Jul 2007 06:44:54 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@openvz.org] wrote:
> | diff -upr linux-2.6.23-rc1-mm1.orig/fs/proc/array.c
> | linux-2.6.23-rc1-mm1-7/fs/proc/array.c
> | --- linux-2.6.23-rc1-mm1.orig/fs/proc/array.c 2007-07-26
> | 16:34:45.000000000 +0400
> | +++ linux-2.6.23-rc1-mm1-7/fs/proc/array.c 2007-07-26
> | 16:36:36.000000000 +0400
> | @@ -77,6 +77,7 @@
> | #include <linux/cpuset.h>
> | #include <linux/rcupdate.h>
> | #include <linux/delayacct.h>
> | +#include <linux/pid_namespace.h>
> |
> | #include <asm/pgtable.h>
> | #include <asm/processor.h>
> | @@ -161,7 +162,9 @@ static inline char *task_state(struct ta
> | struct group_info *group_info;
> | int g;
> | struct fdtable *fdt = NULL;
> | + struct pid_namespace *ns;
> |
> | + ns = current->nsproxy->pid_ns;

> | rcu_read_lock();
> | buffer += sprintf(buffer,
> | "State:\t%s\n"
> | @@ -172,9 +175,12 @@ static inline char *task_state(struct ta
> | "Uid:\t%d\t%d\t%d\t%d\n"
> | "Gid:\t%d\t%d\t%d\t%d\n",
> | get_task_state(p),
> | - p->tgid, p->pid,
> | - pid_alive(p) ? rcu_dereference(p->real_parent)->tgid : 0,
> | - pid_alive(p) && p->ptrace ? rcu_dereference(p->parent)->pid
> | : 0,
> | +   task_tgid_nr_ns(p, ns),
> | + task_pid_nr_ns(p, ns),
> | +   pid_alive(p) ?
> | +   task_ppid_nr_ns(p, ns) : 0,
> | + pid_alive(p) && p->ptrace ?
> | +   task_tgid_nr_ns(rcu_dereference(p->parent), ns) : 0,
>
> Hmm. I think we have this reversed here. We should print the 'tgid' of
> task->real_parent and 'pid' of task->parent.
>

You mean smth like task_tgid_nr_ns(p->real_parent) ? Yes, thanks :)

Subject: Re: [RFC][PATCH 0/15] Pid namespaces
Posted by Pavel Emelianov on Fri, 27 Jul 2007 06:47:18 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> sukadev@us.ibm.com [sukadev@us.ibm.com] wrote:
> | Pavel,
> |
> | We seem to have a memory leak. Its new in this patchset (i.e the
> | following test ran fine on the 2.6.22-rc6-mm1 patchset).
> |
> | To repro: run "pidns_exec ./mypid" in a tight-loop - where mypid.c
> | is:
> |
> | #include <stdio.h>
> | #include <unistd.h>
> |
> | main()
> | {
> | printf("Pid %d, Ppid %d, Pgid %d, Sid %d\n",
> | getpid(), getppid(), getpgid(0), getsid(0));
> | }
> |

> | I ran into OOM in about 30 mins. I am still investigating.
> |
> | BTW, can we include a simple test program like the pidns_exec in this
> | patch-0 for whoever want to play with pidns ?
> |
> | Suka
> |
>
> I think the problem is in create_pid_namespace(). kref_init() sets the
> refcount to 1 and then we do a get_pid_ns() which sets it to 2.
>
> free_nsproxy() frees one of this references, but the other is never
> freed.

That's it! I've switched from my old fast (and bad) reference counting
scheme to new slow (and correct) one, but forgot to remove this "get".
This is the real problem. Thanks :)

> This patch seems to fix the leak. The patch also creates a slab cache
> for the pid_namespace

OK, I'll include this patch. Thanks.

> ---
>
> Create a slab-cache for 'struct pid_namespace' and fix a memory leak
> due to an extra reference in create_pid_namespace().
>
> ---
> kernel/pid.c | 10 ++++++----
> 1 file changed, 6 insertions(+), 4 deletions(-)
>
> Index: lx26-23-rc1-mm1/kernel/pid.c
> ===
> --- lx26-23-rc1-mm1.orig/kernel/pid.c 2007-07-26 20:08:16.000000000 -0700
> +++ lx26-23-rc1-mm1/kernel/pid.c 2007-07-26 22:31:42.000000000 -0700
> @@ -35,6 +35,7 @@
> static struct hlist_head *pid_hash;
> static int pidhash_shift;
> struct pid init_struct_pid = INIT_STRUCT_PID;
> +static struct kmem_cache *pid_ns_cachep;
>
> int pid_max = PID_MAX_DEFAULT;
>
> @@ -525,7 +526,7 @@ static struct pid_namespace *create_pid_
>   struct pid_namespace *ns;
>   int i;
>

```
> - ns = kmalloc(sizeof(struct pid_namespace), GFP_KERNEL);
> + ns = kmem_cache_alloc(pid_ns_cachep, GFP_KERNEL);
>   if (ns == NULL)
>    goto out;
>
> @@ -544,7 +545,6 @@ static struct pid_namespace *create_pid_
>
>   set_bit(0, ns->pidmap[0].page);
>   atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
> - get_pid_ns(ns);
>
>   for (i = 1; i < PIDMAP_ENTRIES; i++) {
>    ns->pidmap[i].page = 0;
> @@ -556,7 +556,7 @@ static struct pid_namespace *create_pid_
>  out_free_map:
>   kfree(ns->pidmap[0].page);
>  out_free:
> - kfree(ns);
> + kmem_cache_free(pid_ns_cachep, ns);
>  out:
>   return ERR_PTR(-ENOMEM);
>  }
> @@ -567,7 +567,7 @@ static void destroy_pid_namespace(struct
>
>   for (i = 0; i < PIDMAP_ENTRIES; i++)
>    kfree(ns->pidmap[i].page);
> - kfree(ns);
> + kmem_cache_free(pid_ns_cachep, ns);
>  }
>
>    struct pid_namespace *copy_pid_ns(unsigned long flags, struct pid_namespace *old_ns)
> @@ -687,4 +687,6 @@ void __init pidmap_init(void)
>   init_pid_ns.pid_cachep = create_pid_cachep(1);
>   if (init_pid_ns.pid_cachep == NULL)
>    panic("Can't create pid_1 cachep\n");
> +
> + pid_ns_cachep = KMEM_CACHE(pid_namespace, SLAB_PANIC);
>  }
>
```


Subject: Re: [PATCH 12/15] Miscelaneous stuff for pid namespaces

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@openvz.org] wrote:
> | This includes
> | * the headers dependency fix
> | * task_child_reaper() correct declaration
> | * fixes for is_global_init() and is_container_init()
> |
> | Maybe this should come before all the other stuff...
> |
> | Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
> |
> | ---
> |
> | include/linux/pid_namespace.h | 4 ++--
> | include/linux/sched.h | 4 ++--
> | kernel/pid.c | 16 ++++++++++++---
> | 3 files changed, 17 insertions(+), 7 deletions(-)
> |
> | diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h
> | linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h
> | --- linux-2.6.23-rc1-mm1.orig/include/linux/pid_namespace.h 2007-07-26
> | 16:34:45.000000000 +0400
> | +++ linux-2.6.23-rc1-mm1-7/include/linux/pid_namespace.h 2007-07-26
> | 16:36:36.000000000 +0400
> | @@ -4,7 +4,6 @@
> | #include <linux/sched.h>
> | #include <linux/mm.h>
> | #include <linux/threads.h>
> | -#include <linux/pid.h>
> | #include <linux/nsproxy.h>
> | #include <linux/kref.h>
> |
> | @@ -46,7 +53,8 @@ static inline struct pid_namespace *task
> |
> | static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
> | {
> | - return init_pid_ns.child_reaper;
> | + BUG_ON(tsk != current);
> | + return tsk->nsproxy->pid_ns->child_reaper;
> | }
> |
> | #endif /* _LINUX_PID_NS_H */
> | diff -upr linux-2.6.23-rc1-mm1.orig/include/linux/sched.h
> | linux-2.6.23-rc1-mm1-7/include/linux/sched.h
> | --- linux-2.6.23-rc1-mm1.orig/include/linux/sched.h 2007-07-26
> | 16:34:45.000000000 +0400

> | +++ linux-2.6.23-rc1-mm1-7/include/linux/sched.h 2007-07-26
> | 16:36:37.000000000 +0400
> | @@ -1277,13 +1366,13 @@ static inline int pid_alive(struct task_
> | *
> | * TODO: We should inline this function after some cleanups in
> | pid_namespace.h
> | */
>
> We can now remove this TODO.

Ok.

> | -extern int is_global_init(struct task_struct *tsk);
> | +extern int is_container_init(struct task_struct *tsk);
> |
> | /*
> | * is_container_init:
> | * check whether in the task is init in its own pid namespace.
> | */
>
> The function header describes is_container_init() but the function
> is actually is_global_init(). The opp is true for the function
> heaer of is_container_init() above.

:)

> | -static inline int is_container_init(struct task_struct *tsk)
> | +static inline int is_global_init(struct task_struct *tsk)
> | {
> | return tsk->pid == 1;
> | }
> | diff -upr linux-2.6.23-rc1-mm1.orig/kernel/pid.c
> | linux-2.6.23-rc1-mm1-7/kernel/pid.c
> | --- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26
> | 16:34:45.000000000 +0400
> | +++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26
> | 16:36:37.000000000 +0400
> | @@ -60,11 +62,21 @@ static inline int mk_pid(struct pid_name
> | };
> | EXPORT_SYMBOL(init_pid_ns);
> |
> | -int is_global_init(struct task_struct *tsk)
> | +int is_container_init(struct task_struct *tsk)
> | {
> | - return tsk == init_pid_ns.child_reaper;
> | + int ret;
>
> Initialize ret = 0 here would save a line of code below :-)

Well, I don't actually like initializing variables right when
they are declared. Even with trivial values :)

> | + struct pid *pid;
> | +
> | + ret = 0;
> | + rcu_read_lock();
> | + pid = task_pid(tsk);
> | + if (pid != NULL && pid->numbers[pid->level].nr == 1)
> | + ret = 1;
> | + rcu_read_unlock();
> | +
> | + return ret;
> | }
> | -EXPORT_SYMBOL(is_global_init);
> | +EXPORT_SYMBOL(is_container_init);
> |
> | /*
> | * Note: disable interrupts while the pidmap_lock is held as an
>

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Oleg Nesterov on Fri, 27 Jul 2007 08:07:58 GMT
View Forum Message <> Reply to Message

On 07/26, Oleg Nesterov wrote:
>
> On 07/26, Pavel Emelyanov wrote:
> >
> > Make task release its namespaces after it has reparented all his
> > children to child_reaper, but before it notifies its parent about
> > its death.
> >
> > The reason to release namespaces after reparenting is that when task
> > exits it may send a signal to its parent (SIGCHLD), but if the parent
> > has already exited its namespaces there will be no way to decide what
> > pid to dever to him - parent can be from different namespace.
> >
> > The reason to release namespace before notifying the parent it that
> > when task sends a SIGCHLD to parent it can call wait() on this taks
> > and release it. But releasing the mnt namespace implies dropping
> > of all the mounts in the mnt namespace and NFS expects the task to
> > have valid sighand pointer.
> >
> > Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
> >

> > ---
> >
> > exit.c | 5 ++++-
> > 1 files changed, 4 insertions(+), 1 deletion(-)
> >
> > diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c
> > linux-2.6.23-rc1-mm1-7/kernel/exit.c
> > --- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26
> > 16:34:45.000000000 +0400
> > +++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26
> > 16:36:37.000000000 +0400
> > @@ -788,6 +804,10 @@ static void exit_notify(struct task_stru
> > BUG_ON(!list_empty(&tsk->children));
> > BUG_ON(!list_empty(&tsk->ptrace_children));
> >
> > + write_unlock_irq(&tasklist_lock);
> > + exit_task_namespaces(tsk);
> > + write_lock_irq(&tasklist_lock);
>
> No.
>
> We "cleared" our ->children/->ptrace_children lists. Now suppose that
> another thread dies, and its forget_original_parent() choose us as a
> new reaper before we re-take tasklist.

Perhaps, we can do something like the patch below. Roland, what do you
think?

We can check PF_EXITING instead of ->exit_state while choosing the new
parent. Note that tasklits_lock acts as a barrier, everyone who takes
tasklist after us (when forget_original_parent() drops it) must see
PF_EXITING.

Oleg.

```
--- t/kernel/exit.c~ 2007-07-27 11:32:21.000000000 +0400
+++ t/kernel/exit.c 2007-07-27 11:59:09.000000000 +0400
@@ -686,11 +686,14 @@ reparent_thread(struct task_struct *p, s
  * the child reaper process (ie "init") in our pid
  * space.
  */
-static void
-forget_original_parent(struct task_struct *father, struct list_head *to_release)
+static void forget_original_parent(struct task_struct *father)
 {
  struct task_struct *p, *reaper = father;
- struct list_head *_p, *_n;
+ struct list_head *ptrace_dead, *_p, *_n;
```

```
+
+ INIT_LIST_HEAD(&ptrace_dead);
+
+ write_lock_irq(&tasklist_lock);

  do {
   reaper = next_thread(reaper);
@@ -698,7 +701,7 @@ forget_original_parent(struct task_struc
    reaper = child_reaper(father);
    break;
   }
- } while (reaper->exit_state);
+ } while (reaper->flags & PF_EXITING);

  /*
   * There are only two places where our children can be:
@@ -736,13 +739,25 @@ forget_original_parent(struct task_struc
    * while it was being traced by us, to be able to see it in wait4.
    */
   if (unlikely(ptrace && p->exit_state == EXIT_ZOMBIE && p->exit_signal == -1))
-   list_add(&p->ptrace_list, to_release);
+   list_add(&p->ptrace_list, &ptrace_dead);
  }
+
  list_for_each_safe(_p, _n, &father->ptrace_children) {
   p = list_entry(_p, struct task_struct, ptrace_list);
   choose_new_parent(p, reaper);
   reparent_thread(p, father, 1);
  }
+
+ write_unlock_irq(&tasklist_lock);
+ BUG_ON(!list_empty(&tsk->children));
+ BUG_ON(!list_empty(&tsk->ptrace_children));
+
+ list_for_each_safe(_p, _n, &ptrace_dead) {
+  list_del_init(_p);
+  t = list_entry(_p, struct task_struct, ptrace_list);
+  release_task(t);
+ }
+
 }

 /*
@@ -753,7 +768,6 @@ static void exit_notify(struct task_stru
 {
  int state;
  struct task_struct *t;
- struct list_head ptrace_dead, *_p, *_n;
```

```
  struct pid *pgrp;

  if (signal_pending(tsk) && !(tsk->signal->flags & SIGNAL_GROUP_EXIT)
@@ -776,8 +790,6 @@ static void exit_notify(struct task_stru
   read_unlock(&tasklist_lock);
  }

- write_lock_irq(&tasklist_lock);
-
  /*
   * This does two things:
   *
@@ -786,12 +798,9 @@ static void exit_notify(struct task_stru
   * as a result of our exiting, and if they have any stopped
   * jobs, send them a SIGHUP and then a SIGCONT.  (POSIX 3.2.2.2)
   */
+ forget_original_parent(tsk);

- INIT_LIST_HEAD(&ptrace_dead);
- forget_original_parent(tsk, &ptrace_dead);
- BUG_ON(!list_empty(&tsk->children));
- BUG_ON(!list_empty(&tsk->ptrace_children));
-
+ write_lock_irq(&tasklist_lock);
  /*
   * Check to see if any process groups have become orphaned
   * as a result of our exiting, and if they have any stopped
@@ -801,9 +810,8 @@ static void exit_notify(struct task_stru
   * and we were the only connection outside, so our pgrp
   * is about to become orphaned.
   */
-
  t = tsk->real_parent;
-
+
  pgrp = task_pgrp(tsk);
  if ((task_pgrp(t) != pgrp) &&
      (task_session(t) == task_session(tsk)) &&
@@ -826,9 +834,8 @@ static void exit_notify(struct task_stru
   * If our self_exec id doesn't match our parent_exec_id then
   * we have changed execution domain as these two values started
   * the same after a fork.
-  *
   */
-
+
  if (tsk->exit_signal != SIGCHLD && tsk->exit_signal != -1 &&
      ( tsk->parent_exec_id != t->self_exec_id  ||
```

```
        tsk->self_exec_id != tsk->parent_exec_id)
@@ -856,12 +863,6 @@ static void exit_notify(struct task_stru

  write_unlock_irq(&tasklist_lock);

- list_for_each_safe(_p, _n, &ptrace_dead) {
-  list_del_init(_p);
-  t = list_entry(_p, struct task_struct, ptrace_list);
-  release_task(t);
- }
-
  /* If the process is dead, release it - nobody will wait for it */
  if (state == EXIT_DEAD)
   release_task(tsk);
```

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Pavel Emelianov on Fri, 27 Jul 2007 08:24:33 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Oleg Nesterov wrote:
>> On 07/26, Pavel Emelyanov wrote:
>>> Make task release its namespaces after it has reparented all his
>>> children to child_reaper, but before it notifies its parent about
>>> its death.
>>>
>>> The reason to release namespaces after reparenting is that when task
>>> exits it may send a signal to its parent (SIGCHLD), but if the parent
>>> has already exited its namespaces there will be no way to decide what
>>> pid to dever to him - parent can be from different namespace.
>>>
>>> The reason to release namespace before notifying the parent it that
>>> when task sends a SIGCHLD to parent it can call wait() on this taks
>>> and release it. But releasing the mnt namespace implies dropping
>>> of all the mounts in the mnt namespace and NFS expects the task to
>>> have valid sighand pointer.
>>>
>>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>>>
>>> ---
>>>
>>> exit.c | 5 ++++-
>>> 1 files changed, 4 insertions(+), 1 deletion(-)
>>>
>>> diff -upr linux-2.6.23-rc1-mm1.orig/kernel/exit.c
>>> linux-2.6.23-rc1-mm1-7/kernel/exit.c
>>> --- linux-2.6.23-rc1-mm1.orig/kernel/exit.c 2007-07-26

>>> 16:34:45.000000000 +0400
>>> +++ linux-2.6.23-rc1-mm1-7/kernel/exit.c 2007-07-26
>>> 16:36:37.000000000 +0400
>>> @@ -788,6 +804,10 @@ static void exit_notify(struct task_stru
>>> BUG_ON(!list_empty(&tsk->children));
>>> BUG_ON(!list_empty(&tsk->ptrace_children));
>>>
>>> + write_unlock_irq(&tasklist_lock);
>>> + exit_task_namespaces(tsk);
>>> + write_lock_irq(&tasklist_lock);
>> No.
>>
>> We "cleared" our ->children/->ptrace_children lists. Now suppose that
>> another thread dies, and its forget_original_parent() choose us as a
>> new reaper before we re-take tasklist.
>
> Perhaps, we can do something like the patch below. Roland, what do you
> think?
>
> We can check PF_EXITING instead of ->exit_state while choosing the new

Heh :) I've came to the same conclusion and now I'm checking for it.
But my patch is much simpler that yours - it just checks for PF_EXITING
in forget_original_parent:

--- ./kernel/exit.c.exitfix 2007-07-27 12:13:25.000000000 +0400
+++ ./kernel/exit.c 2007-07-27 12:15:35.000000000 +0400
@@ -712,7 +712,7 @@ forget_original_parent(struct task_struc
    reaper = task_child_reaper(father);
    break;
   }
- } while (reaper->exit_state);
+ } while (reaper->flags & PF_EXITING);

  /*
   * There are only two places where our children can be:

> parent. Note that tasklits_lock acts as a barrier, everyone who takes
> tasklist after us (when forget_original_parent() drops it) must see
> PF_EXITING.
>
> Oleg.
>
> --- t/kernel/exit.c~ 2007-07-27 11:32:21.000000000 +0400
> +++ t/kernel/exit.c 2007-07-27 11:59:09.000000000 +0400
> @@ -686,11 +686,14 @@ reparent_thread(struct task_struct *p, s
>   * the child reaper process (ie "init") in our pid

```
>   * space.
>   */
> -static void
> -forget_original_parent(struct task_struct *father, struct list_head *to_release)
> +static void forget_original_parent(struct task_struct *father)
>  {
>   struct task_struct *p, *reaper = father;
> - struct list_head *_p, *_n;
> + struct list_head *ptrace_dead, *_p, *_n;
> +
> + INIT_LIST_HEAD(&ptrace_dead);
> +
> + write_lock_irq(&tasklist_lock);
>   do {
>    reaper = next_thread(reaper);
> @@ -698,7 +701,7 @@ forget_original_parent(struct task_struc
>     reaper = child_reaper(father);
>     break;
>    }
> - } while (reaper->exit_state);
> + } while (reaper->flags & PF_EXITING);
>
>   /*
>    * There are only two places where our children can be:
> @@ -736,13 +739,25 @@ forget_original_parent(struct task_struc
>     * while it was being traced by us, to be able to see it in wait4.
>     */
>    if (unlikely(ptrace && p->exit_state == EXIT_ZOMBIE && p->exit_signal == -1))
> -   list_add(&p->ptrace_list, to_release);
> +   list_add(&p->ptrace_list, &ptrace_dead);
>   }
> +
>   list_for_each_safe(_p, _n, &father->ptrace_children) {
>    p = list_entry(_p, struct task_struct, ptrace_list);
>    choose_new_parent(p, reaper);
>    reparent_thread(p, father, 1);
>   }
> +
> + write_unlock_irq(&tasklist_lock);
> + BUG_ON(!list_empty(&tsk->children));
> + BUG_ON(!list_empty(&tsk->ptrace_children));
> +
> + list_for_each_safe(_p, _n, &ptrace_dead) {
> +  list_del_init(_p);
> +  t = list_entry(_p, struct task_struct, ptrace_list);
> +  release_task(t);
> + }
> +
```

```
>  }
>
>  /*
> @@ -753,7 +768,6 @@ static void exit_notify(struct task_stru
>  {
>   int state;
>   struct task_struct *t;
> - struct list_head ptrace_dead, *_p, *_n;
>   struct pid *pgrp;
>
>   if (signal_pending(tsk) && !(tsk->signal->flags & SIGNAL_GROUP_EXIT)
> @@ -776,8 +790,6 @@ static void exit_notify(struct task_stru
>    read_unlock(&tasklist_lock);
>   }
>
> - write_lock_irq(&tasklist_lock);
> -
>   /*
>    * This does two things:
>    *
> @@ -786,12 +798,9 @@ static void exit_notify(struct task_stru
>    * as a result of our exiting, and if they have any stopped
>    * jobs, send them a SIGHUP and then a SIGCONT.  (POSIX 3.2.2.2)
>    */
> + forget_original_parent(tsk);
>
> - INIT_LIST_HEAD(&ptrace_dead);
> - forget_original_parent(tsk, &ptrace_dead);
> - BUG_ON(!list_empty(&tsk->children));
> - BUG_ON(!list_empty(&tsk->ptrace_children));
> -
> + write_lock_irq(&tasklist_lock);
>   /*
>    * Check to see if any process groups have become orphaned
>    * as a result of our exiting, and if they have any stopped
> @@ -801,9 +810,8 @@ static void exit_notify(struct task_stru
>    * and we were the only connection outside, so our pgrp
>    * is about to become orphaned.
>    */
> -
>   t = tsk->real_parent;
> -
> +
>   pgrp = task_pgrp(tsk);
>   if ((task_pgrp(t) != pgrp) &&
>       (task_session(t) == task_session(tsk)) &&
> @@ -826,9 +834,8 @@ static void exit_notify(struct task_stru
>    * If our self_exec id doesn't match our parent_exec_id then
```

>   * we have changed execution domain as these two values started
>   * the same after a fork.
> - *
>   */
> -
> +
>   if (tsk->exit_signal != SIGCHLD && tsk->exit_signal != -1 &&
>   (tsk->parent_exec_id != t->self_exec_id ||
>   tsk->self_exec_id != tsk->parent_exec_id)
> @@ -856,12 +863,6 @@ static void exit_notify(struct task_stru
>
>   write_unlock_irq(&tasklist_lock);
>
> - list_for_each_safe(_p, _n, &ptrace_dead) {
> - list_del_init(_p);
> - t = list_entry(_p, struct task_struct, ptrace_list);
> - release_task(t);
> - }
> -
>   /* If the process is dead, release it - nobody will wait for it */
>   if (state == EXIT_DEAD)
>   release_task(tsk);
>
>

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Oleg Nesterov on Fri, 27 Jul 2007 08:35:33 GMT
View Forum Message <> Reply to Message

On 07/27, Pavel Emelyanov wrote:
>
> Oleg Nesterov wrote:
> >
> >Perhaps, we can do something like the patch below. Roland, what do you
> >think?
> >
> >We can check PF_EXITING instead of ->exit_state while choosing the new
>
> Heh :) I've came to the same conclusion and now I'm checking for it.
> But my patch is much simpler that yours - it just checks for PF_EXITING
> in forget_original_parent:
>
> --- ./kernel/exit.c.exitfix 2007-07-27 12:13:25.000000000 +0400
> +++ ./kernel/exit.c 2007-07-27 12:15:35.000000000 +0400
> @@ -712,7 +712,7 @@ forget_original_parent(struct task_struc
>   reaper = task_child_reaper(father);
>   break;

>   }
> - } while (reaper->exit_state);
> + } while (reaper->flags & PF_EXITING);

Yes, other changes are just cleanups. They just move some code from exit_notify
to forget_original_parent(). It is a bit silly to declare ptrace_dead in
exit_notify(), take tasklist, pass ptrace_dead to forget_original_parent(),
unlock-lock-unlock tasklist, and then use ptrace_dead.

Oleg.

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Pavel Emelianov on Fri, 27 Jul 2007 08:37:21 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/27, Pavel Emelyanov wrote:
>> Oleg Nesterov wrote:
>>> Perhaps, we can do something like the patch below. Roland, what do you
>>> think?
>>>
>>> We can check PF_EXITING instead of ->exit_state while choosing the new
>> Heh :) I've came to the same conclusion and now I'm checking for it.
>> But my patch is much simpler that yours - it just checks for PF_EXITING
>> in forget_original_parent:
>>
>> --- ./kernel/exit.c.exitfix 2007-07-27 12:13:25.000000000 +0400
>> +++ ./kernel/exit.c 2007-07-27 12:15:35.000000000 +0400
>> @@ -712,7 +712,7 @@ forget_original_parent(struct task_struc
>>   reaper = task_child_reaper(father);
>>   break;
>>   }
>> - } while (reaper->exit_state);
>> + } while (reaper->flags & PF_EXITING);
>
> Yes, other changes are just cleanups. They just move some code from exit_notify
> to forget_original_parent(). It is a bit silly to declare ptrace_dead in
> exit_notify(), take tasklist, pass ptrace_dead to forget_original_parent(),
> unlock-lock-unlock tasklist, and then use ptrace_dead.

:) Ok. Thanks, I then will check with your patch. If you don't mind
I will carry it together with this set.

> Oleg.

Thanks,
Pavel

Subject: Re: [PATCH 11/15] Signal semantics
Posted by Oleg Nesterov on Fri, 27 Jul 2007 12:30:12 GMT
View Forum Message <> Reply to Message

Damn. I don't have time to read these patches today (will try tomorrow),
but when I glanced at this patch yesterday I had some suspicions...

On 07/26, Pavel Emelyanov wrote:
>
> +++ linux-2.6.23-rc1-mm1-7/kernel/signal.c 2007-07-26
> 16:36:37.000000000 +0400
> @@ -323,6 +325,9 @@ static int collect_signal(int sig, struc
> if (first) {
> list_del_init(&first->list);
> copy_siginfo(info, &first->info);
> + if (first->flags & SIGQUEUE_CINIT)
> + kinfo->flags |= KERN_SIGINFO_CINIT;
> +
>
> [...snip...]
>
> @@ -1852,7 +1950,7 @@ relock:
> * within that pid space. It can of course get signals from
> * its parent pid space.
> */
> - if (current == task_child_reaper(current))
> + if (kinfo.flags & KERN_SIGINFO_CINIT)
> continue;

I think the whole idea is broken, it assumes the sender put something into
"struct sigqueue".

Suppose that /sbin/init has no handler for (say) SIGTERM, and we send this
signal from the same namespace. send_signal() sets SIGQUEUE_CINIT, but it
is lost because __group_complete_signal() silently "converts" sig_fatal()
signals to SIGKILL using sigaddset().

> +static void encode_sender_info(struct task_struct *t, struct sigqueue *q)
> +{
> + /*
> + * If sender (i.e 'current') and receiver have the same active
> + * pid namespace and the receiver is the container-init, set the
> + * SIGQUEUE_CINIT flag. This tells the container-init that the
> + * signal originated in its own namespace and so it can choose
> + * to ignore the signal.
> + *
> + * If the receiver is the container-init of a pid namespace,
> + * but the sender is from an ancestor pid namespace, the
> + * container-init cannot ignore the signal. So clear the

> + * SIGQUEUE_CINIT flag in this case.
> + *
> + * Also, if the sender does not have a pid_t in the receiver's
> + * active pid namespace, set si_pid to 0 and pretend it originated
> + * from the kernel.
> + */
> + if (pid_ns_equal(t)) {
> + if (is_container_init(t)) {
> +   q->flags |= SIGQUEUE_CINIT;

Ironically, this change carefully preserves the bug we already have :)

This doesn't protect init from "bad" signal if we send it to sub-thread
of init. Actually, this make the behaviour a bit worse compared to what
we currently have. Currently, at least the main init's thread survives
if we send SIGKILL to sub-thread.

> static int send_signal(int sig, struct siginfo *info, struct task_struct *t,
>   struct sigpending *signals)
> {
> @@ -710,6 +781,7 @@ static int send_signal(int sig, struct s
>   copy_siginfo(&q->info, info);
>   break;
>   }
> + encode_sender_info(t, q);

We still send the signal if __sigqueue_alloc() fails. In that case, the
dequeued siginfo won't have SIGQUEUE_CINIT/KERN_SIGINFO_CINIT, not good.

> @@ -1158,6 +1232,13 @@ static int kill_something_info(int sig,
>
>   read_lock(&tasklist_lock);
>   for_each_process(p) {
> +   /*
> +   * System-wide signals apply only to the sender's
> +   * pid namespace, unless issued from init_pid_ns.
> +   */
> +   if (!task_visible_in_pid_ns(p, my_ns))
> +   continue;
> +
>   if (p->pid > 1 && p->tgid != current->tgid) {

This "p->pid > 1" check should die.

> +static int deny_signal_to_container_init(struct task_struct *tsk, int sig)
> +{
> + /*
> + * If receiver is the container-init of sender and signal is SIGKILL

> + * reject it right-away. If signal is any other one, let the container
> + * init decide (in get_signal_to_deliver()) whether to handle it or
> + * ignore it.
> + */
> + if (is_container_init(tsk) && (sig == SIGKILL) && pid_ns_equal(tsk))
> + return -EPERM;
> +
> + return 0;
> +}
> +
> /*
> * Bad permissions for sending the signal
> */
> @@ -545,6 +584,10 @@ static int check_kill_permission(int sig
> && !capable(CAP_KILL))
> return error;
>
> + error = deny_signal_to_container_init(t, sig);
> + if (error)
> + return error;

Hm. Could you explain this change? Why do we need a special check for SIGKILL?

(What about ptrace_attach() btw? If it is possible to send a signal to init
from the "parent" namespace, perhaps it makes sense to allow ptracing as
well).

Oleg.

Subject: Re: [PATCH 11/15] Signal semantics
Posted by Pavel Emelianov on Fri, 27 Jul 2007 13:38:19 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> Damn. I don't have time to read these patches today (will try tomorrow),

Oh, that's OK. I was about to send the set to Andrew only the next week.

This patch is the most strange one and is to be discussed a lot.

We try to do the following two things:
1. signals going from the namespace, that the target task doesn't
see must be seen as SI_KERNEL if siginfo is allocated;
2. signals to init of any namespace must be allowed to send from
one of the parent namespaces only. From child namespace, init
needs only those, that it's ready to handle (SIGCHLD).

As far as I understand Suka's approach (it's his patch, so I may
be not 100% correct - it's better to wait for his comments) he is
trying to carry the information about the signal up to the
get_signal_to_deliver().

As far as the first issue is concerned, the solution is obvious -
all the "calculations" can be done at the beginning of sending the
signal, but the second issue is a bit more complicated and I have
no good ideas of how to solve this :(yet.

Thanks,
Pavel

> but when I glanced at this patch yesterday I had some suspicions...
>
> On 07/26, Pavel Emelyanov wrote:
>> +++ linux-2.6.23-rc1-mm1-7/kernel/signal.c 2007-07-26
>> 16:36:37.000000000 +0400
>> @@ -323,6 +325,9 @@ static int collect_signal(int sig, struc
>> if (first) {
>>   list_del_init(&first->list);
>>   copy_siginfo(info, &first->info);
>> + if (first->flags & SIGQUEUE_CINIT)
>> +   kinfo->flags |= KERN_SIGINFO_CINIT;
>> +
>>
>> [...snip...]
>>
>> @@ -1852,7 +1950,7 @@ relock:
>>   * within that pid space. It can of course get signals from
>>   * its parent pid space.
>>   */
>> - if (current == task_child_reaper(current))
>> + if (kinfo.flags & KERN_SIGINFO_CINIT)
>>   continue;
>
> I think the whole idea is broken, it assumes the sender put something into
> "struct sigqueue".

Yup. That's the problem. It seems to me that the only way to handle init's
signals is to check for permissions in the sending path.

> Suppose that /sbin/init has no handler for (say) SIGTERM, and we send this
> signal from the same namespace. send_signal() sets SIGQUEUE_CINIT, but it
> is lost because __group_complete_signal() silently "converts" sig_fatal()
> signals to SIGKILL using sigaddset().
>

```
>> +static void encode_sender_info(struct task_struct *t, struct sigqueue *q)
>> +{
>> + /*
>> +  * If sender (i.e 'current') and receiver have the same active
>> +  * pid namespace and the receiver is the container-init, set the
>> +  * SIGQUEUE_CINIT flag. This tells the container-init that the
>> +  * signal originated in its own namespace and so it can choose
>> +  * to ignore the signal.
>> +  *
>> +  * If the receiver is the container-init of a pid namespace,
>> +  * but the sender is from an ancestor pid namespace, the
>> +  * container-init cannot ignore the signal. So clear the
>> +  * SIGQUEUE_CINIT flag in this case.
>> +  *
>> +  * Also, if the sender does not have a pid_t in the receiver's
>> +  * active pid namespace, set si_pid to 0 and pretend it originated
>> +  * from the kernel.
>> +  */
>> + if (pid_ns_equal(t)) {
>> +  if (is_container_init(t)) {
>> +   q->flags |= SIGQUEUE_CINIT;
>
> Ironically, this change carefully preserves the bug we already have :)
>
> This doesn't protect init from "bad" signal if we send it to sub-thread
> of init. Actually, this make the behaviour a bit worse compared to what
> we currently have. Currently, at least the main init's thread survives
> if we send SIGKILL to sub-thread.
>
>> static int send_signal(int sig, struct siginfo *info, struct task_struct *t,
>>    struct sigpending *signals)
>> {
>> @@ -710,6 +781,7 @@ static int send_signal(int sig, struct s
>>    copy_siginfo(&q->info, info);
>>    break;
>>   }
>> +  encode_sender_info(t, q);
>
> We still send the signal if __sigqueue_alloc() fails. In that case, the
> dequeued siginfo won't have SIGQUEUE_CINIT/KERN_SIGINFO_CINIT, not good.
>
>> @@ -1158,6 +1232,13 @@ static int kill_something_info(int sig,
>>
>>   read_lock(&tasklist_lock);
>>   for_each_process(p) {
>> +   /*
>> +    * System-wide signals apply only to the sender's
>> +    * pid namespace, unless issued from init_pid_ns.
```

>> +   */
>> +   if (!task_visible_in_pid_ns(p, my_ns))
>> +   continue;
>> +
>>   if (p->pid > 1 && p->tgid != current->tgid) {
>
> This "p->pid > 1" check should die.
>
>> +static int deny_signal_to_container_init(struct task_struct *tsk, int sig)
>> +{
>> + /*
>> + * If receiver is the container-init of sender and signal is SIGKILL
>> + * reject it right-away. If signal is any other one, let the container
>> + * init decide (in get_signal_to_deliver()) whether to handle it or
>> + * ignore it.
>> + */
>> + if (is_container_init(tsk) && (sig == SIGKILL) && pid_ns_equal(tsk))
>> + return -EPERM;
>> +
>> + return 0;
>> +}
>> +
>> /*
>> * Bad permissions for sending the signal
>> */
>> @@ -545,6 +584,10 @@ static int check_kill_permission(int sig
>>   && !capable(CAP_KILL))
>>   return error;
>>
>> + error = deny_signal_to_container_init(t, sig);
>> + if (error)
>> + return error;
>
> Hm. Could you explain this change? Why do we need a special check for SIGKILL?
>
>
> (What about ptrace_attach() btw? If it is possible to send a signal to init
> from the "parent" namespace, perhaps it makes sense to allow ptracing as
> well).

ptracing of tasks fro different namespaces is not possible at all, since
strace utility determines the fork()-ed child pid from the parent's eax
register, which would contain the pid value as this parent sees his child.
But if the strace is in different namespace - it won't be able to find
this child with the pid value from parent's eax.

Maybe it's worth disabling cross-namespaces ptracing...

>
> Oleg.
>
>

Subject: Re: [PATCH 9/15] Move alloc_pid() after the namespace is cloned
Posted by Oleg Nesterov on Fri, 27 Jul 2007 15:10:54 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:
>
> This is a fix for Sukadev's patch that moved the alloc_pid() call from
> do_fork() into copy_process().

... and this patch changes almost every line from Sukadev's patch.
Sorry gents, but isn't it better to ask Andrew to drop that patch
(which is quite useless by itself), and send a new one which incorporates
all necessary changes? Imho, it would be much easier to understand.

> @@ -1406,7 +1422,13 @@ long do_fork(unsigned long clone_flags,
> if (!IS_ERR(p)) {
>   struct completion vfork;
>
> - nr = pid_nr(task_pid(p));
> + /*
> +   * this is enough to call pid_nr_ns here, but this if
> +   * improves optimisation of regular fork()
> +   */
> + nr = (clone_flags & CLONE_NEWPID) ?
> +   task_pid_nr_ns(p, current->nsproxy->pid_ns) :
> +   task_pid_vnr(p);

Shouldn't we do the same for CLONE_PARENT_SETTID in copy_process() ?
Otherwise *parent_tidptr may have a wrong value which doesn't match
to what fork() returns.

Oleg.

Subject: Re: [PATCH 11/15] Signal semantics
Posted by serue on Fri, 27 Jul 2007 19:59:43 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
> Pavel Emelianov [xemul@openvz.org] wrote:
> | Oleg Nesterov wrote:

> | >Damn. I don't have time to read these patches today (will try tomorrow),
> |
> | Oh, that's OK. I was about to send the set to Andrew only the next week.
> |
> | This patch is the most strange one and is to be discussed a lot.
> |
> | We try to do the following two things:
> | 1. signals going from the namespace, that the target task doesn't
> | see must be seen as SI_KERNEL if siginfo is allocated;
> | 2. signals to init of any namespace must be allowed to send from
> | one of the parent namespaces only. From child namespace, init
> | needs only those, that it's ready to handle (SIGCHLD).
>
> Yes.
>
> |
> | As far as I understand Suka's approach (it's his patch, so I may
> | be not 100% correct - it's better to wait for his comments) he is
> | trying to carry the information about the signal up to the
> | get_signal_to_deliver().
> |
> | As far as the first issue is concerned, the solution is obvious -
> | all the "calculations" can be done at the beginning of sending the
> | signal, but the second issue is a bit more complicated and I have
> | no good ideas of how to solve this :(yet.
>
> Even I am looking for a better approach.
>
> |
> | Thanks,
> | Pavel
> |
> | >but when I glanced at this patch yesterday I had some suspicions...
> | >
> | >On 07/26, Pavel Emelyanov wrote:
> | >>+++ linux-2.6.23-rc1-mm1-7/kernel/signal.c 2007-07-26
> | >>16:36:37.000000000 +0400
> | >>@@ -323,6 +325,9 @@ static int collect_signal(int sig, struc
> | >> if (first) {
> | >> list_del_init(&first->list);
> | >> copy_siginfo(info, &first->info);
> | >>+ if (first->flags & SIGQUEUE_CINIT)
> | >>+   kinfo->flags |= KERN_SIGINFO_CINIT;
> | >>+
> | >>
> | >>[...snip...]
> | >>
> | >>@@ -1852,7 +1950,7 @@ relock:

> | >> * within that pid space. It can of course get signals from
> | >> * its parent pid space.
> | >> */
> | >>- if (current == task_child_reaper(current))
> | >>+ if (kinfo.flags & KERN_SIGINFO_CINIT)
> | >> continue;
> | >
> | >I think the whole idea is broken, it assumes the sender put something into
> | >"struct sigqueue".
> |
> | Yup. That's the problem. It seems to me that the only way to handle init's
> | signals is to check for permissions in the sending path.
>
> We can check permissions in the sending path - and in fact we do check for
> SIGKILL case (deny_signal_to_container_init() below).
>
> But the receiver knows/decides whether or not the signal is wanted/not. No ?
>
> Are you saying we should check/special case all fatal signals ?
>
> |
> | >Suppose that /sbin/init has no handler for (say) SIGTERM, and we send this
> | >signal from the same namespace. send_signal() sets SIGQUEUE_CINIT, but it
> | >is lost because __group_complete_signal() silently "converts" sig_fatal()
> | >signals to SIGKILL using sigaddset().
>
> Yes, I should have called it out, but this patch currently assumes /sbin/init
> (or container-init) has a handler for the fatal signals like SIGTERM and has
> a check for SIGKILL (in deny_signal_to_container_init() - as Oleg noted below).
>
> Still looking for better ways to implement.
>
> | >
> | >>+static void encode_sender_info(struct task_struct *t, struct sigqueue *q)
> | >>+{
> | >>+ /*
> | >>+ * If sender (i.e 'current') and receiver have the same active
> | >>+ * pid namespace and the receiver is the container-init, set the
> | >>+ * SIGQUEUE_CINIT flag. This tells the container-init that the
> | >>+ * signal originated in its own namespace and so it can choose
> | >>+ * to ignore the signal.
> | >>+ *
> | >>+ * If the receiver is the container-init of a pid namespace,
> | >>+ * but the sender is from an ancestor pid namespace, the
> | >>+ * container-init cannot ignore the signal. So clear the
> | >>+ * SIGQUEUE_CINIT flag in this case.
> | >>+ *
> | >>+ * Also, if the sender does not have a pid_t in the receiver's

> | >>+ * active pid namespace, set si_pid to 0 and pretend it originated
> | >>+ * from the kernel.
> | >>+ */
> | >>+ if (pid_ns_equal(t)) {
> | >>+ if (is_container_init(t)) {
> | >>+   q->flags |= SIGQUEUE_CINIT;
> | >
> | >Ironically, this change carefully preserves the bug we already have :)
> | >
> | >This doesn't protect init from "bad" signal if we send it to sub-thread
> | >of init. Actually, this make the behaviour a bit worse compared to what
> | >we currently have. Currently, at least the main init's thread survives
> | >if we send SIGKILL to sub-thread.
>
> Do you mean "init's main thread" ? But doesn't SIGKILL to any thread kill
> the entire process ?
>
> | >
> | >>static int send_signal(int sig, struct siginfo *info, struct task_struct
> | >>*t,
> | >>   struct sigpending *signals)
> | >>{
> | >>@@ -710,6 +781,7 @@ static int send_signal(int sig, struct s
> | >>   copy_siginfo(&q->info, info);
> | >>   break;
> | >> }
> | >>+ encode_sender_info(t, q);
> | >
> | >We still send the signal if __sigqueue_alloc() fails. In that case, the
> | >dequeued siginfo won't have SIGQUEUE_CINIT/KERN_SIGINFO_CINIT, not good.
>
> Yes.
>
> | >
> | >>@@ -1158,6 +1232,13 @@ static int kill_something_info(int sig,
> | >>
> | >> read_lock(&tasklist_lock);
> | >> for_each_process(p) {
> | >>+   /*
> | >>+   * System-wide signals apply only to the sender's
> | >>+   * pid namespace, unless issued from init_pid_ns.
> | >>+   */
> | >>+   if (!task_visible_in_pid_ns(p, my_ns))
> | >>+   continue;
> | >>+
> | >>   if (p->pid > 1 && p->tgid != current->tgid) {
> | >
> | >This "p->pid > 1" check should die.

> | >
>
> Ok.
>
> | >>+static int deny_signal_to_container_init(struct task_struct *tsk, int
> | >>sig)
> | >>+{
> | >>+ /*
> | >>+ * If receiver is the container-init of sender and signal is SIGKILL
> | >>+ * reject it right-away. If signal is any other one, let the
> | >>container
> | >>+ * init decide (in get_signal_to_deliver()) whether to handle it or
> | >>+ * ignore it.
> | >>+ */
> | >>+ if (is_container_init(tsk) && (sig == SIGKILL) && pid_ns_equal(tsk))
> | >>+ return -EPERM;
> | >>+
> | >>+ return 0;
> | >>+}
> | >>+
> | >>/*
> | >> * Bad permissions for sending the signal
> | >> */
> | >>@@ -545,6 +584,10 @@ static int check_kill_permission(int sig
> | >>   && !capable(CAP_KILL))
> | >> return error;
> | >>
> | >>+ error = deny_signal_to_container_init(t, sig);
> | >>+ if (error)
> | >>+ return error;
> | >
> | >Hm. Could you explain this change? Why do we need a special check for
> | >SIGKILL?
>
> As you pointed out above, SIGKILL goes through the __group_complete_signal()/
> sigaddset() path and bypasses/loses the KERN_SIGINFO_CINIT flag. Other
> sig_fatal() signals take this path too, but we assume for now, container-init
> has a handler.
>
>
> | >
> | >
> | >(What about ptrace_attach() btw? If it is possible to send a signal to init
> | > from the "parent" namespace, perhaps it makes sense to allow ptracing as
> | > well).
> |
> | ptracing of tasks fro different namespaces is not possible at all, since
> | strace utility determines the fork()-ed child pid from the parent's eax

> | register, which would contain the pid value as this parent sees his child.
> | But if the strace is in different namespace - it won't be able to find
> | this child with the pid value from parent's eax.
> |
> | Maybe it's worth disabling cross-namespaces ptracing...
>
> I think so too. Its probably not a serious limitation ?

Several people think we will implement 'namespace entering' through a
ptrace hack, where maybe the admin ptraces the init in a child pidns,
makes it fork, and makes the child execute what it wants (i.e. ps -ef).

You're talking about killing that functionality?

-serge

Subject: Re: [PATCH 11/15] Signal semantics
Posted by Sukadev Bhattiprolu on Fri, 27 Jul 2007 20:23:37 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn [serue@us.ibm.com] wrote:
| Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
| > Pavel Emelianov [xemul@openvz.org] wrote:
| > | Oleg Nesterov wrote:
| > | >(What about ptrace_attach() btw? If it is possible to send a signal to init
| > | > from the "parent" namespace, perhaps it makes sense to allow ptracing as
| > | > well).
| > |
| > | ptracing of tasks fro different namespaces is not possible at all, since
| > | strace utility determines the fork()-ed child pid from the parent's eax
| > | register, which would contain the pid value as this parent sees his child.
| > | But if the strace is in different namespace - it won't be able to find
| > | this child with the pid value from parent's eax.
| > |
| > | Maybe it's worth disabling cross-namespaces ptracing...
| >
| > I think so too. Its probably not a serious limitation ?
|
| Several people think we will implement 'namespace entering' through a
| ptrace hack, where maybe the admin ptraces the init in a child pidns,
| makes it fork, and makes the child execute what it wants (i.e. ps -ef).
|
| You're talking about killing that functionality?

No. I was only thinking in terms of debugging container init and missed
the namespace entering part.

Pavel, I am not sure I understand your comment about being unable to
ptrace() a child ns.

BTW, I am able to gdb a process (incl container-init) from parent ns now.

|
| -serge

Subject: Re: [PATCH 5/15] Introduce struct upid
Posted by Oleg Nesterov on Sun, 29 Jul 2007 09:50:35 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:
>
> --- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
> +++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
> @@ -40,15 +40,21 @@ enum pid_type
> * processes.
> */
>
> -struct pid
> -{
> - atomic_t count;
> +struct upid {
> /* Try to keep pid_chain in the same cacheline as nr for find_pid */
> int nr;
> + struct pid_namespace *ns;
> struct hlist_node pid_chain;
> +};
> +
> +struct pid
> +{
> + atomic_t count;
> /* lists of tasks that use this pid */
> struct hlist_head tasks[PIDTYPE_MAX];
> struct rcu_head rcu;
> + int level;
> + struct upid numbers[1];
> };

Well. Definitely, the kernel can't be compiled with this patch applied,
this seems to be against the rules...

So. The task has a single (PIDTYPE_MAX) pid no matter how many namespaces
can see it, and "struct pid" has an array of numbers for each namespace.

Still I can't understand why do we need upid->ns, can't we kill it?

Suppose we add "struct pid_namespace *parent_ns" to "struct pid_namespace",
init_pid_ns.parent_ns == NULL.

Now,

```
 struct upid {
  int nr;
  struct hlist_node pid_chain;
 };

 struct pid
 {
  atomic_t count;
  struct hlist_head tasks[PIDTYPE_MAX];
  struct rcu_head rcu;
  struct pid_namespace *active_ns;
  struct upid numbers[0];
 };
```

We populate pid->numbers in "reverse" order, so that pid->numbers[0] lives
in pid->active_ns.

Now, for example,

```
 void free_pid(struct pid *pid)
 {
  struct pid_namespace *ns;
  unsigned long flags;
  int i;

  spin_lock_irqsave(&pidmap_lock, flags);
  for (i = 0, ns = pid->active_ns; ns; i++, ns = ns->parent_ns)
   hlist_del_rcu(&pid->numbers[i].pid_chain);
  spin_unlock_irqrestore(&pidmap_lock, flags);

  for (i = 0, ns = pid->active_ns; ns; i++, ns = ns->parent_ns)
   free_pidmap(ns, pid->numbers[i].nr);

  call_rcu(&pid->rcu, delayed_put_pid);
 }
```

Possible?

Oleg.

Subject: Re: [PATCH 6/15] Make alloc_pid(), free_pid() and put_pid() work with

On 07/26, Pavel Emelyanov wrote:
>
> -struct pid *alloc_pid(void)
> +struct pid *alloc_pid(struct pid_namespace *ns)

Why? We have the only caller, copy_process(), ns == task_active_pid_ns()
always.

> {
>   struct pid *pid;
>   enum pid_type type;
> - int nr = -1;
> - struct pid_namespace *ns;
> + int i, nr;
> + struct pid_namespace *tmp;
>
> - ns = task_active_pid_ns(current);
>   pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
>   if (!pid)
>   goto out;
>
> - nr = alloc_pidmap(ns);
> - if (nr < 0)
> - goto out_free;
> + tmp = ns;
> + for (i = ns->level; i >= 0; i--) {
> + nr = alloc_pidmap(tmp);
> + if (nr < 0)
> +   goto out_free;
> +
> + pid->numbers[i].nr = nr;
> + pid->numbers[i].ns = tmp;
> + tmp = tmp->parent;

Hm... There is no ->parent in "struct pid_namespace", and this
patch doesn't add it.

> + if (ns != &init_pid_ns)
> + get_pid_ns(ns);

Q: put_pid() checks "ns != &init_pid_ns" as well, this is just
an optimization, yes? Perhaps we can move this check into
get_pid_ns/put_pid_ns.

We are doing get_pid_ns() only for the "top namespace"... I guess

this can work if pid_namespace does get_pid_ns() on its ->parent.
This patch looks incomplete.

Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Sun, 29 Jul 2007 10:39:47 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:
>
> @@ -895,6 +915,7 @@ fastcall NORET_TYPE void do_exit(long co
> {
> struct task_struct *tsk = current;
> int group_dead;
> + struct pid_namespace *pid_ns = tsk->nsproxy->pid_ns;
>
> profile_task_exit(tsk);
>
> @@ -905,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
> if (unlikely(!tsk->pid))
> panic("Attempted to kill the idle task!");
> if (unlikely(tsk == task_child_reaper(tsk))) {
> - if (task_active_pid_ns(tsk) != &init_pid_ns)
> - task_active_pid_ns(tsk)->child_reaper =
> - init_pid_ns.child_reaper;
> + if (pid_ns != &init_pid_ns) {
> + zap_pid_ns_processes(pid_ns);
> + pid_ns->child_reaper = init_pid_ns.child_reaper;
> + }
> else
> panic("Attempted to kill init!");

No, no, this is wrong. Yes, the current code is buggy too, I'll send
the fix.

I think this code should be moved below under the "if (group_dead)",
and we should use tsk->group_leader.

> +void zap_pid_ns_processes(struct pid_namespace *pid_ns)
> +{
> + int i;
> + int nr;
> + int nfree;
> + int options = WNOHANG|WEXITED|__WALL;
> +
> +repeat:

```
> + /*
> +  * We know pid == 1 is terminating. Find remaining pid_ts
> +  * in the namespace, signal them and then wait for them
> +  * exit.
> +  */
> + nr = next_pidmap(pid_ns, 1);
> + while (nr > 0) {
> +  kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
> +  nr = next_pidmap(pid_ns, nr);
> + }
> +
> + nr = next_pidmap(pid_ns, 1);
> + while (nr > 0) {
> +  do_wait(nr, options, NULL, NULL, NULL);
```

When the first child of init exits, it sends SIGCHLD. After that,
do_wait() will never sleep, so we are doing a busy-wait loop.
Not good, especially when we have a niced child, can livelock.

```
> +  nr = next_pidmap(pid_ns, nr);
> + }
> +
> + nfree = 0;
> + for (i = 0; i < PIDMAP_ENTRIES; i++)
> +  nfree += atomic_read(&pid_ns->pidmap[i].nr_free);
> +
> + /*
> +  * If pidmap has entries for processes other than 0 and 1, retry.
> +  */
> + if (nfree < (BITS_PER_PAGE * PIDMAP_ENTRIES - 2))
> +  goto repeat;
```

This doesn't look right.

Suppose that some "struct pid" was pinned from the parent namespace.
In that case zap_pid_ns_processes() will burn CPU until put_pid(), bad.

I think we can rely on forget_original_child() and do something like
this:

```
 zap_active_ns_processes(void)
 {
  // kill all tasks in our ns and below
  kill(-1, SIGKILL);

  do {
   clear_thread_flag(TIF_SIGPENDING);
  } while (wait(NULL) != -ECHLD);
```

}

Oleg.

Subject: Re: [PATCH 11/15] Signal semantics
Posted by Oleg Nesterov on Sun, 29 Jul 2007 11:23:34 GMT
View Forum Message <> Reply to Message

On 07/27, sukadev@us.ibm.com wrote:
>
> Pavel Emelianov [xemul@openvz.org] wrote:
> | Oleg Nesterov wrote:
> | >>
> | >>@@ -1852,7 +1950,7 @@ relock:
> | >> * within that pid space. It can of course get signals from
> | >> * its parent pid space.
> | >> */
> | >>- if (current == task_child_reaper(current))
> | >>+ if (kinfo.flags & KERN_SIGINFO_CINIT)
> | >> continue;
> | >
> | >I think the whole idea is broken, it assumes the sender put something into
> | >"struct sigqueue".
> |
> | Yup. That's the problem. It seems to me that the only way to handle init's
> | signals is to check for permissions in the sending path.
>
> We can check permissions in the sending path - and in fact we do check for
> SIGKILL case (deny_signal_to_container_init() below).
>
> But the receiver knows/decides whether or not the signal is wanted/not. No ?

I can't understand your question. Yes, this is what we are doing currently,
but this is broken by this patch.

> Are you saying we should check/special case all fatal signals ?
>
> |
> | >Suppose that /sbin/init has no handler for (say) SIGTERM, and we send this
> | >signal from the same namespace. send_signal() sets SIGQUEUE_CINIT, but it
> | >is lost because __group_complete_signal() silently "converts" sig_fatal()
> | >signals to SIGKILL using sigaddset().
>
> Yes, I should have called it out, but this patch currently assumes /sbin/init
> (or container-init) has a handler for the fatal signals like SIGTERM

Changelog says nothing about that. And in that case we don't need any complications

except a) deny_signal_to_container_init() (should be named deny_SIGKILL_to_container_init)
and b) "cross-namespace signals must have si_code == SI_KERNEL".

I don't know whether this limitation (/sbin/init must install the handler
for each fatal signal) acceptable or not.

However, we should also take care about sig_kernel_stop() signals, and please
note that it is not possible to install a handler for SIGSTOP.

> | >>+static void encode_sender_info(struct task_struct *t, struct sigqueue *q)
> | >>+{
> | >>+ if (pid_ns_equal(t)) {
> | >>+ if (is_container_init(t)) {
> | >>+   q->flags |= SIGQUEUE_CINIT;
> | >
> | >Ironically, this change carefully preserves the bug we already have :)
> | >
> | >This doesn't protect init from "bad" signal if we send it to sub-thread
> | >of init. Actually, this make the behaviour a bit worse compared to what
> | >we currently have. Currently, at least the main init's thread survives
> | >if we send SIGKILL to sub-thread.
>
> Do you mean "init's main thread" ?

Yes.

> But doesn't SIGKILL to any thread kill
> the entire process ?

It should, but it doesn't if it was sent to init's sub-thread, exactly
because of child_reaper() check in get_signal_to_deliver().

> | >>+   error = deny_signal_to_container_init(t, sig);
> | >>+   if (error)
> | >>+   return error;
> | >
> | >Hm. Could you explain this change? Why do we need a special check for
> | >SIGKILL?
>
> As you pointed out above, SIGKILL goes through the __group_complete_signal()/
> sigaddset() path and bypasses/loses the KERN_SIGINFO_CINIT flag. Other
> sig_fatal() signals take this path too, but we assume for now, container-init
> has a handler.

No, SIGKILL doesn't bypasses send_signal(). IOW, if other parts of this patch
were correct, we don't need this change. If init has a handler, we don't neeed
other parts.

> | >(What about ptrace_attach() btw? If it is possible to send a signal to init
> | > from the "parent" namespace, perhaps it makes sense to allow ptracing as
> | > well).
> |
> | ptracing of tasks fro different namespaces is not possible at all, since
> | strace utility determines the fork()-ed child pid from the parent's eax
> | register, which would contain the pid value as this parent sees his child.
> | But if the strace is in different namespace - it won't be able to find
> | this child with the pid value from parent's eax.
> |
> | Maybe it's worth disabling cross-namespaces ptracing...
>
> I think so too. Its probably not a serious limitation ?

My question was not clear, sorry. And I was confused because I had a false
impression that ptrace_attach() was already changed to use is_container_init().

Afaics, the cross-namespaces ptracing should work (modulo fork() problem
pointed out by Pavel), and probably it is useful.

But we should fix ptrace_attach(), it should not be possible to do PTRACE_ATTACH
to /sbin/init from the _same_ namespace.

Oleg.

Subject: Re: [PATCH 3/15] kern_siginfo helper
Posted by Oleg Nesterov on Sun, 29 Jul 2007 11:40:00 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:
>
> TODO: This is more an exploratory patch and modifies only interfaces
> necessary to implement correct signal semantics in pid namespaces.
>
> If the approach is feasible, we could consistently use 'kern_siginfo'
> in other signal interfaces and possibly in 'struct sigqueue'.
>
> We could modify dequeue_signal() to directly work with 'kern_siginfo'
> and remove dequeue_signal_kern_info().

Well... I know, it is very easy to blame somebody else's patch, and probably
my taste is not good...

But honestly, I personally think this approach is a horror, and any alternative
is better :)

I'd rather change dequeue_signal() so that it takes "struct sigqueue *"

parameter instead of "siginfo_t *", or add a new "int *flags".

OK, this doesn't work anyway, we should do something different. Perhaps
just do all checks on sender's side.

It is a bit strange that this patch is 3/15, and the rest bits in 11/15,
not very convenient for the review.

Oleg.

Subject: Re: [PATCH 7/15] Helpers to obtain pid numbers
Posted by Oleg Nesterov on Sun, 29 Jul 2007 12:08:57 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:
>
> --- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
> +++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
> @@ -83,12 +92,34 @@ extern void FASTCALL(detach_pid(struct t
>
> extern struct pid *alloc_pid(struct pid_namespace *ns);
> extern void FASTCALL(free_pid(struct pid *pid));
> +
> +/*
> + * the helpers to get the pid's id seen from different namespaces
> + *
> + * pid_nr()   : global id, i.e. the id seen from the init namespace;

This looks a bit strange to me, but perhaps this is just matter of taste.
I think pid_nr(pid) should be pid_nr_ns(pid, current->nsproxy->pid_ns),
this is imho much closer to the current meaning. I won't persist though.

> +pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
> +{
> + pid_t nr = 0;
> + if (pid && ns->level <= pid->level)
> + nr = pid->numbers[ns->level].nr;
> + return nr;
> +}

I am not sure I understand the "ns->level <= pid->level" check. Isn't it
a bug to use a "wrong" namespace here? In that case BUG_ON() looks better.

If ns could be wrong, "ns->level <= pid->level" is not enough, we should
also check "pid->numbers[ns->level].ns == ns", no?

Oleg.

Subject: Re: [PATCH 8/15] Helpers to find the task by its numerical ids
Posted by Oleg Nesterov on Sun, 29 Jul 2007 12:39:07 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:
>
> +#define find_pid(pid) find_pid_ns(pid, &init_pid_ns)

Again, I think find_pid() should use current's active ns, not
init_pid_ns. Just grep for find_pid/find_task_by_pid.

> --- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
> +++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
> @@ -204,19 +221,20 @@ static void delayed_put_pid(struct rcu_h
> goto out;
> }
>
> -struct pid * fastcall find_pid(int nr)
> +struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
> {
> struct hlist_node *elem;
> - struct pid *pid;
> + struct upid *pnr;
> +
> + hlist_for_each_entry_rcu(pnr, elem,
> + &pid_hash[pid_hashfn(nr, ns)], pid_chain)
> + if (pnr->nr == nr && pnr->ns == ns)
                              ^^^^^^^^^^^^^

Aha, that is why we need upid->ns.

I am a bit surprised we don't move the global pid_hash into the
"struct pid_namespace", this could speedup the search, and we
don't need upid->ns.

> -struct pid *find_ge_pid(int nr)
> +struct pid *find_ge_pid(int nr, struct pid_namespace *ns)
> {
> struct pid *pid;
>
> do {
> - pid = find_pid(nr);
> + pid = find_pid_ns(nr, ns);
> if (pid)
> break;
> - nr = next_pidmap(task_active_pid_ns(current), nr);
> + nr = next_pidmap(ns, nr);
> } while (nr > 0);
>
> return pid;

This means we should fix the caller, next_tgid(), but this is done
in 15/15.

Oleg.

Subject: Re: [PATCH 15/15] Hooks over the code to show correct values to user
Posted by Oleg Nesterov on Sun, 29 Jul 2007 14:29:59 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:
>
> int
> kill_proc(pid_t pid, int sig, int priv)
> {
> - return kill_proc_info(sig, __si_special(priv), pid);
> + int ret;
> +
> + rcu_read_lock();
> + ret = kill_pid_info(sig, __si_special(priv), find_pid(pid));
> + rcu_read_unlock();
> + return ret;
> }

I think this is wrong. kill_proc() should behave the same as kill_proc_info(),
so this change is not needed. With this patch they use different namespaces
to find the task, this is not consistent.

(sadly, this patch is huge, very difficult to review).

Oleg.

Subject: Re: [PATCH 10/15] Make each namespace has its own proc tree
Posted by Oleg Nesterov on Sun, 29 Jul 2007 15:56:50 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:
>
> static int proc_get_sb(struct file_system_type *fs_type,
> int flags, const char *dev_name, void *data, struct vfsmount *mnt)
> {
> [...snip...]
>
> + if (!sb->s_root) {
> + sb->s_flags = flags;

> + err = proc_fill_super(sb);
> + if (err) {
> +   up_write(&sb->s_umount);
> +   deactivate_super(sb);
> +   return err;
> + }
> +
> + ei = PROC_I(sb->s_root->d_inode);
> + if (!ei->pid) {
> +   rcu_read_lock();
> +   ei->pid = get_pid(find_pid_ns(1, ns));

Where do we "put" this pid?

> void release_task(struct task_struct * p)
> {
> + struct pid *pid;
> struct task_struct *leader;
> int zap_leader;
> repeat:
> @@ -160,6 +161,20 @@ repeat:
> write_lock_irq(&tasklist_lock);
> ptrace_unlink(p);
> BUG_ON(!list_empty(&p->ptrace_list) ||
> !list_empty(&p->ptrace_children));
> + /*
> + * we have to keep this pid till proc_flush_task() to make
> + * it possible to flush all dentries holding it. pid will
> + * be put ibidem
> + *
> + * however if the pid belogs to init namespace only, we can
> + * optimize this out
> + */
> + pid = task_pid(p);
> + if (pid->level != 0)
> + get_pid(pid);
> + else
> + pid = NULL;
> +
> __exit_signal(p);
>
> /*
> @@ -184,7 +199,8 @@ repeat:
> }
>
> write_unlock_irq(&tasklist_lock);
> - proc_flush_task(p, NULL);
> + proc_flush_task(p, pid);

> + put_pid(pid);

Oh, this is not nice...

Look, proc_flush_task() doesn't need pid, it can use active pid_ns
to iterate the ->parent chain and do proc_flush_task_mnt(), and it
can check "ns->child_reaper == current" for mntput(), yes?

So, can't we do instead

```
 release_task()
 {
  struct pid_namespace *my_ns = ...; // get it from pid;

  ...

  write_unlock_irq(&tasklist_lock);

  // __exit_signal()->...->put_pid() drops the reference,
  // but this ns should still be valid because /proc itself
  // holds a reference to it, even if we are /sbin/init.
  proc_flush_task(p, my_ns);
  ...
 }
```

No?

Oleg.

Subject: Re: [PATCH 10/15] Make each namespace has its own proc tree
Posted by Oleg Nesterov on Sun, 29 Jul 2007 17:02:53 GMT
View Forum Message <> Reply to Message

On 07/29, Oleg Nesterov wrote:
>
> Look, proc_flush_task() doesn't need pid, it can use active pid_ns

Oops. Yes, proc_flush_task() doesn't use ->numbers[].nr, but it should?

proc_flush_task_mnt() uses task->pid to flush /proc/$pid, this looks wrong,
we should use ->number[ns->level].nr, no?

Oleg.

Subject: Re: [PATCH 5/15] Introduce struct upid

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> --- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
>> +++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
>> @@ -40,15 +40,21 @@ enum pid_type
>> * processes.
>> */
>>
>> -struct pid
>> -{
>> - atomic_t count;
>> +struct upid {
>> /* Try to keep pid_chain in the same cacheline as nr for find_pid */
>> int nr;
>> + struct pid_namespace *ns;
>> struct hlist_node pid_chain;
>> +};
>> +
>> +struct pid
>> +{
>> + atomic_t count;
>> /* lists of tasks that use this pid */
>> struct hlist_head tasks[PIDTYPE_MAX];
>> struct rcu_head rcu;
>> + int level;
>> + struct upid numbers[1];
>> };
>
> Well. Definitely, the kernel can't be compiled with this patch applied,
> this seems to be against the rules...

Yes. U forgot to mention, that this patchset is git-bisect-not-safe :)
I sent the safe split earlier, but it was harder to make and understand,
so I decided not to waste the time and sent a badly-split set just to get
comments about the approach. The ways a big patch is split wouldn't affect
the comments about the ideas, bugs, etc.

> So. The task has a single (PIDTYPE_MAX) pid no matter how many namespaces
> can see it, and "struct pid" has an array of numbers for each namespace.
>
> Still I can't understand why do we need upid->ns, can't we kill it?
> Suppose we add "struct pid_namespace *parent_ns" to "struct pid_namespace",
> init_pid_ns.parent_ns == NULL.

We already have it :)

> Now,
>
> struct upid {
>   int nr;
>   struct hlist_node pid_chain;
> };
>
> struct pid
> {
>   atomic_t count;
>   struct hlist_head tasks[PIDTYPE_MAX];
>   struct rcu_head rcu;
>   struct pid_namespace *active_ns;
>   struct upid numbers[0];
> };
>
> We populate pid->numbers in "reverse" order, so that pid->numbers[0] lives
> in pid->active_ns.
>
> Now, for example,
>
> void free_pid(struct pid *pid)
> {
>   struct pid_namespace *ns;
>   unsigned long flags;
>   int i;
>
>   spin_lock_irqsave(&pidmap_lock, flags);
>   for (i = 0, ns = pid->active_ns; ns; i++, ns = ns->parent_ns)
>   hlist_del_rcu(&pid->numbers[i].pid_chain);
>   spin_unlock_irqrestore(&pidmap_lock, flags);
>
>   for (i = 0, ns = pid->active_ns; ns; i++, ns = ns->parent_ns)
>   free_pidmap(ns, pid->numbers[i].nr);
>
>   call_rcu(&pid->rcu, delayed_put_pid);
> }
>
> Possible?

Possible, but how will
struct pid *find_pid_nr_ns(int nr, struct pid_namespace *ns);
look then? The only way (I see) is to make

hlist_for_each_entry (upid, ...)
    if (upid->nr == nr && upid->ns == ns)
            return container_of(upid, struct pid, ...)

> Oleg.

Thanks,
Pavel

Subject: Re: [PATCH 6/15] Make alloc_pid(), free_pid() and put_pid() work with struct upid
Posted by Pavel Emelianov on Mon, 30 Jul 2007 06:03:05 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> -struct pid *alloc_pid(void)
>> +struct pid *alloc_pid(struct pid_namespace *ns)
>
> Why? We have the only caller, copy_process(), ns == task_active_pid_ns()
> always.

task_active_pid_ns() by newly created task, not the current! That's why
we need to pass something to alloc_pid() to find this new namespace.
Task or namespace itself - is the matter of choice - I selected the
most obvious argument :)

>> {
>>   struct pid *pid;
>>   enum pid_type type;
>> - int nr = -1;
>> - struct pid_namespace *ns;
>> + int i, nr;
>> + struct pid_namespace *tmp;
>>
>> - ns = task_active_pid_ns(current);
>>   pid = kmem_cache_alloc(ns->pid_cachep, GFP_KERNEL);
>>   if (!pid)
>>   goto out;
>>
>> - nr = alloc_pidmap(ns);
>> - if (nr < 0)
>> - goto out_free;
>> + tmp = ns;
>> + for (i = ns->level; i >= 0; i--) {
>> + nr = alloc_pidmap(tmp);
>> + if (nr < 0)
>> +   goto out_free;
>> +
>> + pid->numbers[i].nr = nr;
>> + pid->numbers[i].ns = tmp;

>> + tmp = tmp->parent;
>
> Hm... There is no ->parent in "struct pid_namespace", and this
> patch doesn't add it.

Parent is added in another patch - 12/15. I will split it better
when sending to Andrew - patches will be smaller and bisect-safe.

>> + if (ns != &init_pid_ns)
>> + get_pid_ns(ns);
>
> Q: put_pid() checks "ns != &init_pid_ns" as well, this is just
> an optimization, yes? Perhaps we can move this check into

It is :)

> get_pid_ns/put_pid_ns.

I think you're right.

> We are doing get_pid_ns() only for the "top namespace"... I guess
> this can work if pid_namespace does get_pid_ns() on its ->parent.
> This patch looks incomplete.

Yes. This set is not well split, sorry. I wanted to get comments about the
approach, bugs, etc (I have already mentioned this in another letter...)

> Oleg.
>
>

Thanks,
Pavel.

Subject: Re: [PATCH 3/15] kern_siginfo helper
Posted by Pavel Emelianov on Mon, 30 Jul 2007 06:07:46 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> TODO: This is more an exploratory patch and modifies only interfaces
>> necessary to implement correct signal semantics in pid namespaces.
>>
>> If the approach is feasible, we could consistently use 'kern_siginfo'
>> in other signal interfaces and possibly in 'struct sigqueue'.
>>
>> We could modify dequeue_signal() to directly work with 'kern_siginfo'

>> and remove dequeue_signal_kern_info().
>
> Well... I know, it is very easy to blame somebody else's patch, and probably
> my taste is not good...
>
> But honestly, I personally think this approach is a horror, and any alternative
> is better :)
>
> I'd rather change dequeue_signal() so that it takes "struct sigqueue *"
> parameter instead of "siginfo_t *", or add a new "int *flags".
>
> OK, this doesn't work anyway, we should do something different. Perhaps
> just do all checks on sender's side.

Yes. Signal handling in namespaces turned out to be the most complicated
part of the set. I start thinking to drop this part till we have the "core"
in -mm tree. Suka, what do you think?

> It is a bit strange that this patch is 3/15, and the rest bits in 11/15,
> not very convenient for the review.

Well, I thought that a split like
1. patches for kernel to prepare it for the set
2. the set itself
is the best to review. Maybe I was wrong, but how to make this then?
E.g. I have a MS_KERNOUNT patch, but its changes are used *much* later.

> Oleg.
>
>

Thanks,
Pavel

Subject: Re: [PATCH 7/15] Helpers to obtain pid numbers
Posted by Pavel Emelianov on Mon, 30 Jul 2007 06:11:25 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> --- linux-2.6.23-rc1-mm1.orig/include/linux/pid.h 2007-07-26 16:34:45.000000000 +0400
>> +++ linux-2.6.23-rc1-mm1-7/include/linux/pid.h 2007-07-26 16:36:37.000000000 +0400
>> @@ -83,12 +92,34 @@ extern void FASTCALL(detach_pid(struct t
>>
>> extern struct pid *alloc_pid(struct pid_namespace *ns);
>> extern void FASTCALL(free_pid(struct pid *pid));
>> +

>> +/*
>> + * the helpers to get the pid's id seen from different namespaces
>> + *
>> + * pid_nr()   : global id, i.e. the id seen from the init namespace;
>
> This looks a bit strange to me, but perhaps this is just matter of taste.
> I think pid_nr(pid) should be pid_nr_ns(pid, current->nsproxy->pid_ns),
> this is imho much closer to the current meaning. I won't persist though.

pid_nr, find_task_by_pid and all other stuff, that existed in the kernel
before the set are intended to work with global ids only (just as it was
before the set).

>> +pid_t pid_nr_ns(struct pid *pid, struct pid_namespace *ns)
>> +{
>> + pid_t nr = 0;
>> + if (pid && ns->level <= pid->level)
>> + nr = pid->numbers[ns->level].nr;
>> + return nr;
>> +}
>
> I am not sure I understand the "ns->level <= pid->level" check. Isn't it
> a bug to use a "wrong" namespace here? In that case BUG_ON() looks better.

Yes, that's a check for bad namespace passed.

> If ns could be wrong, "ns->level <= pid->level" is not enough, we should
> also check "pid->numbers[ns->level].ns == ns", no?

Yes, we should, and you're right in that we must have BUG_ON() here.

> Oleg.
>
>

Thanks,
Pavel

Subject: Re: [PATCH 8/15] Helpers to find the task by its numerical ids
Posted by Pavel Emelianov on Mon, 30 Jul 2007 06:15:22 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> +#define find_pid(pid) find_pid_ns(pid, &init_pid_ns)
>
> Again, I think find_pid() should use current's active ns, not

> init_pid_ns. Just grep for find_pid/find_task_by_pid.
>
>> --- linux-2.6.23-rc1-mm1.orig/kernel/pid.c 2007-07-26 16:34:45.000000000 +0400
>> +++ linux-2.6.23-rc1-mm1-7/kernel/pid.c 2007-07-26 16:36:37.000000000 +0400
>> @@ -204,19 +221,20 @@ static void delayed_put_pid(struct rcu_h
>> goto out;
>> }
>>
>> -struct pid * fastcall find_pid(int nr)
>> +struct pid * fastcall find_pid_ns(int nr, struct pid_namespace *ns)
>> {
>> struct hlist_node *elem;
>> - struct pid *pid;
>> + struct upid *pnr;
>> +
>> + hlist_for_each_entry_rcu(pnr, elem,
>> + &pid_hash[pid_hashfn(nr, ns)], pid_chain)
>> + if (pnr->nr == nr && pnr->ns == ns)
>> ^^^^^^^^^^^^
> Aha, that is why we need upid->ns.

That's it :)

> I am a bit surprised we don't move the global pid_hash into the
> "struct pid_namespace", this could speedup the search, and we
> don't need upid->ns.

Hm... Worth thinking about, but this hash itself is large enough and
its size depends on the node's number of pages, so we'll have

1. either to make per-namespace hash (much) smaller;
2. or to give (too) many memory for it.

>> -struct pid *find_ge_pid(int nr)
>> +struct pid *find_ge_pid(int nr, struct pid_namespace *ns)
>> {
>> struct pid *pid;
>>
>> do {
>> - pid = find_pid(nr);
>> + pid = find_pid_ns(nr, ns);
>> if (pid)
>> break;
>> - nr = next_pidmap(task_active_pid_ns(current), nr);
>> + nr = next_pidmap(ns, nr);
>> } while (nr > 0);
>>
>> return pid;

>
> This means we should fix the caller, next_tgid(), but this is done
> in 15/15.

Sorry :)

> Oleg.
>
>

Thank,
Pavel

Subject: Re: [PATCH 9/15] Move alloc_pid() after the namespace is cloned
Posted by Pavel Emelianov on Mon, 30 Jul 2007 06:17:10 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> This is a fix for Sukadev's patch that moved the alloc_pid() call from
>> do_fork() into copy_process().
>
> ... and this patch changes almost every line from Sukadev's patch.

It does. My bad :(I have reviewed Suka's patch badly and was sure it
puts the alloc_pid() right where we need this.

> Sorry gents, but isn't it better to ask Andrew to drop that patch
> (which is quite useless by itself), and send a new one which incorporates
> all necessary changes? Imho, it would be much easier to understand.

Hm... Maybe it's better to ask him to fold these patches together?

>> @@ -1406,7 +1422,13 @@ long do_fork(unsigned long clone_flags,
>> if (!IS_ERR(p)) {
>>   struct completion vfork;
>>
>> - nr = pid_nr(task_pid(p));
>> + /*
>> +   * this is enough to call pid_nr_ns here, but this if
>> +   * improves optimisation of regular fork()
>> +   */
>> + nr = (clone_flags & CLONE_NEWPID) ?
>> +   task_pid_nr_ns(p, current->nsproxy->pid_ns) :
>> +   task_pid_vnr(p);
>
> Shouldn't we do the same for CLONE_PARENT_SETTID in copy_process() ?

> Otherwise *parent_tidptr may have a wrong value which doesn't match
> to what fork() returns.

Oops. We should. Thanks :)

> Oleg.

Thanks,
Pavel

Subject: Re: [PATCH 10/15] Make each namespace has its own proc tree
Posted by Pavel Emelianov on Mon, 30 Jul 2007 06:43:37 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> static int proc_get_sb(struct file_system_type *fs_type,
>> int flags, const char *dev_name, void *data, struct vfsmount *mnt)
>> {
>> [...snip...]
>>
>> + if (!sb->s_root) {
>> + sb->s_flags = flags;
>> + err = proc_fill_super(sb);
>> + if (err) {
>> +   up_write(&sb->s_umount);
>> +   deactivate_super(sb);
>> +   return err;
>> + }
>> +
>> + ei = PROC_I(sb->s_root->d_inode);
>> + if (!ei->pid) {
>> +   rcu_read_lock();
>> +   ei->pid = get_pid(find_pid_ns(1, ns));
>
> Where do we "put" this pid?

In proc_delete_inode()

>> void release_task(struct task_struct * p)
>> {
>> + struct pid *pid;
>> struct task_struct *leader;
>> int zap_leader;
>> repeat:
>> @@ -160,6 +161,20 @@ repeat:
>> write_lock_irq(&tasklist_lock);

>> ptrace_unlink(p);
>> BUG_ON(!list_empty(&p->ptrace_list) ||
>> !list_empty(&p->ptrace_children));
>> + /*
>> + * we have to keep this pid till proc_flush_task() to make
>> + * it possible to flush all dentries holding it. pid will
>> + * be put ibidem
>> + *
>> + * however if the pid belogs to init namespace only, we can
>> + * optimize this out
>> + */
>> + pid = task_pid(p);
>> + if (pid->level != 0)
>> + get_pid(pid);
>> + else
>> + pid = NULL;
>> +
>> __exit_signal(p);
>>
>> /*
>> @@ -184,7 +199,8 @@ repeat:
>> }
>>
>> write_unlock_irq(&tasklist_lock);
>> - proc_flush_task(p, NULL);
>> + proc_flush_task(p, pid);
>> + put_pid(pid);
>
> Oh, this is not nice...
>
> Look, proc_flush_task() doesn't need pid, it can use active pid_ns

It cannot. By the time release_task() is called the task is already
exit_task_namespaces()-ed :(

> to iterate the ->parent chain and do proc_flush_task_mnt(), and it
> can check "ns->child_reaper == current" for mntput(), yes?
>
> So, can't we do instead
>
> release_task()
> {
>   struct pid_namespace *my_ns = ...; // get it from pid;

So, that's what you mean. Well, that's sounds reasonable. Thanks.

>   ...
>

> write_unlock_irq(&tasklist_lock);
>
> // __exit_signal()->...->put_pid() drops the reference,
> // but this ns should still be valid because /proc itself
> // holds a reference to it, even if we are /sbin/init.
> proc_flush_task(p, my_ns);
> ...
> }
>
> No?

Yes. Thanks!

> Oleg.

Pavel

Subject: Re: [PATCH 10/15] Make each namespace has its own proc tree
Posted by Pavel Emelianov on Mon, 30 Jul 2007 06:45:46 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/29, Oleg Nesterov wrote:
>> Look, proc_flush_task() doesn't need pid, it can use active pid_ns
>
> Oops. Yes, proc_flush_task() doesn't use ->numbers[].nr, but it should?
>
> proc_flush_task_mnt() uses task->pid to flush /proc/$pid, this looks wrong,
> we should use ->number[ns->level].nr, no?

Indeed. Hmm, why then pids were released during my tests...
Looks like we do still need struct pid for proc_flush_task().

> Oleg.

Thanks,
Pavel

Subject: Re: [PATCH 15/15] Hooks over the code to show correct values to user
Posted by Pavel Emelianov on Mon, 30 Jul 2007 06:49:10 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> int

>> kill_proc(pid_t pid, int sig, int priv)
>> {
>> - return kill_proc_info(sig, __si_special(priv), pid);
>> + int ret;
>> +
>> + rcu_read_lock();
>> + ret = kill_pid_info(sig, __si_special(priv), find_pid(pid));
>> + rcu_read_unlock();
>> + return ret;
>> }
>
> I think this is wrong. kill_proc() should behave the same as kill_proc_info(),
> so this change is not needed. With this patch they use different namespaces
> to find the task, this is not consistent.

Actually, callers of this use tsk->pid (global pid) as an argument, so
find_vpid() might return wrong value.

> (sadly, this patch is huge, very difficult to review).

This is *very* huge, but all it does is just replace tsk->pid, find_task_by_pid,
pd_nr, etc with appropriate task_pid_nr(), pid_nr_ns() and find_task_by_vpid() etc.

> Oleg.

Thanks,
Pavel

Subject: Re: [PATCH 11/15] Signal semantics
Posted by Pavel Emelianov on Mon, 30 Jul 2007 09:31:44 GMT
View Forum Message <> Reply to Message

[snip]

>> | Maybe it's worth disabling cross-namespaces ptracing...
>>
>> I think so too. Its probably not a serious limitation ?
>
> Several people think we will implement 'namespace entering' through a
> ptrace hack, where maybe the admin ptraces the init in a child pidns,

Why not implement namespace entering w/o any hacks? :)

> makes it fork, and makes the child execute what it wants (i.e. ps -ef).
>
> You're talking about killing that functionality?

No. We're talking about disabling the things that are not supposed
to work at all.

> -serge
>

Subject: Re: [PATCH 11/15] Signal semantics
Posted by Pavel Emelianov on Mon, 30 Jul 2007 09:34:57 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Serge E. Hallyn [serue@us.ibm.com] wrote:
> | Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
> | > Pavel Emelianov [xemul@openvz.org] wrote:
> | > | Oleg Nesterov wrote:
> | > | >(What about ptrace_attach() btw? If it is possible to send a signal to init
> | > | > from the "parent" namespace, perhaps it makes sense to allow ptracing as
> | > | > well).
> | > |
> | > | ptracing of tasks fro different namespaces is not possible at all, since
> | > | strace utility determines the fork()-ed child pid from the parent's eax
> | > | register, which would contain the pid value as this parent sees his child.
> | > | But if the strace is in different namespace - it won't be able to find
> | > | this child with the pid value from parent's eax.
> | > |
> | > | Maybe it's worth disabling cross-namespaces ptracing...
> | >
> | > I think so too. Its probably not a serious limitation ?
> |
> | Several people think we will implement 'namespace entering' through a
> | ptrace hack, where maybe the admin ptraces the init in a child pidns,
> | makes it fork, and makes the child execute what it wants (i.e. ps -ef).
> |
> | You're talking about killing that functionality?
>
> No. I was only thinking in terms of debugging container init and missed
> the namespace entering part.
>
> Pavel, I am not sure I understand your comment about being unable to
> ptrace() a child ns.

Ok. We have a task with pid 100, which tries to clone the new namespace.
This task fork-s and we have a new task with a couple of pids (101, 1).
Then this "init" forks again and we have the third task with pids (102, 2).
The problem is that when the 3rd task appears the return value from fork(),
that is - the new task's pid as it is seen by it's parent (2nd task), will
go to eax register (for i386) and it will be 2! But the prtaces from the

initial namespace cannot address this task with pid 2.

> BTW, I am able to gdb a process (incl container-init) from parent ns now.

Debugging separate processes is possible, but when this task starts forking
with namespaces creation - this becomes impossible.

> |
> | -serge
>

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Pavel Emelianov on Mon, 30 Jul 2007 11:56:50 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> @@ -895,6 +915,7 @@ fastcall NORET_TYPE void do_exit(long co
>> {
>> struct task_struct *tsk = current;
>> int group_dead;
>> + struct pid_namespace *pid_ns = tsk->nsproxy->pid_ns;
>>
>> profile_task_exit(tsk);
>>
>> @@ -905,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
>> if (unlikely(!tsk->pid))
>> panic("Attempted to kill the idle task!");
>> if (unlikely(tsk == task_child_reaper(tsk))) {
>> - if (task_active_pid_ns(tsk) != &init_pid_ns)
>> - task_active_pid_ns(tsk)->child_reaper =
>> - init_pid_ns.child_reaper;
>> + if (pid_ns != &init_pid_ns) {
>> + zap_pid_ns_processes(pid_ns);
>> + pid_ns->child_reaper = init_pid_ns.child_reaper;
>> + }
>> else
>> panic("Attempted to kill init!");
>
> No, no, this is wrong. Yes, the current code is buggy too, I'll send
> the fix.
>
> I think this code should be moved below under the "if (group_dead)",
> and we should use tsk->group_leader.
>
>> +void zap_pid_ns_processes(struct pid_namespace *pid_ns)
>> +{

>> + int i;
>> + int nr;
>> + int nfree;
>> + int options = WNOHANG|WEXITED|__WALL;
>> +
>> +repeat:
>> + /*
>> + * We know pid == 1 is terminating. Find remaining pid_ts
>> + * in the namespace, signal them and then wait for them
>> + * exit.
>> + */
>> + nr = next_pidmap(pid_ns, 1);
>> + while (nr > 0) {
>> + kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
>> + nr = next_pidmap(pid_ns, nr);
>> + }
>> +
>> + nr = next_pidmap(pid_ns, 1);
>> + while (nr > 0) {
>> + do_wait(nr, options, NULL, NULL, NULL);
>
> When the first child of init exits, it sends SIGCHLD. After that,
> do_wait() will never sleep, so we are doing a busy-wait loop.
> Not good, especially when we have a niced child, can livelock.
>
>> + nr = next_pidmap(pid_ns, nr);
>> + }
>> +
>> + nfree = 0;
>> + for (i = 0; i < PIDMAP_ENTRIES; i++)
>> + nfree += atomic_read(&pid_ns->pidmap[i].nr_free);
>> +
>> + /*
>> + * If pidmap has entries for processes other than 0 and 1, retry.
>> + */
>> + if (nfree < (BITS_PER_PAGE * PIDMAP_ENTRIES - 2))
>> + goto repeat;
>
> This doesn't look right.
>
> Suppose that some "struct pid" was pinned from the parent namespace.
> In that case zap_pid_ns_processes() will burn CPU until put_pid(), bad.

Nope. struct pid can be pinned, but the pidmap "fingerprint" cannot.
So as soon as the release_task() is called the pidmap becomes free and
we can proceed.

However I agree with the "burn CPU" issue - wait must sleep if needed.

> I think we can rely on forget_original_child() and do something like
> this:
>
> zap_active_ns_processes(void)
> {
>   // kill all tasks in our ns and below
>   kill(-1, SIGKILL);

That would be too slow to walk through all the tasks in a node searching
for a couple of them we need. fing_ge_pid() looks better to me.

>   do {
>   clear_thread_flag(TIF_SIGPENDING);
>   } while (wait(NULL) != -ECHLD);
> }
>
> Oleg.
>
>

Thanks,
Pavel

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Mon, 30 Jul 2007 15:44:57 GMT
View Forum Message <> Reply to Message

On 07/30, Pavel Emelyanov wrote:
>
> Oleg Nesterov wrote:
> >>+
> >>+ nfree = 0;
> >>+ for (i = 0; i < PIDMAP_ENTRIES; i++)
> >>+ nfree += atomic_read(&pid_ns->pidmap[i].nr_free);
> >>+
> >>+ /*
> >>+ * If pidmap has entries for processes other than 0 and 1, retry.
> >>+ */
> >>+ if (nfree < (BITS_PER_PAGE * PIDMAP_ENTRIES - 2))
> >>+ goto repeat;
> >
> >This doesn't look right.
> >
> >Suppose that some "struct pid" was pinned from the parent namespace.
> >In that case zap_pid_ns_processes() will burn CPU until put_pid(), bad.
>

> Nope. struct pid can be pinned, but the pidmap "fingerprint" cannot.

Heh. It was specially designed this way, but I managed to forget.

You are right, thanks for correcting me.

> So as soon as the release_task() is called the pidmap becomes free and
> we can proceed.

Well, it doesn't matter, but strictly speaking this is not true.
release_task()->detach_pid(PIDTYPE_PID) doesn't necessary free pidmap,
it could be "used" by other tasks as PGID/SID.

> However I agree with the "burn CPU" issue - wait must sleep if needed.
>
> >I think we can rely on forget_original_child() and do something like
> >this:
> >
> > zap_active_ns_processes(void)
> > {
> > // kill all tasks in our ns and below
> > kill(-1, SIGKILL);
>
> That would be too slow to walk through all the tasks in a node searching
> for a couple of them we need. fing_ge_pid() looks better to me.

OK. I personally dislike the "retry" logic (and it is not safe if we want
wait() to actually sleep waiting for the child), but we can do the "kill"
loop under tasklist_lock.

In any case, I don't think we should use next_pid() for wait(), or check
pidmap[].nr_free,

> > do {
> > clear_thread_flag(TIF_SIGPENDING);
> > } while (wait(NULL) != -ECHLD);
> > }

just wait for any child until -ECHLD. After that we can return safely.

Oleg.

Subject: Re: [PATCH 9/15] Move alloc_pid() after the namespace is cloned
Posted by Sukadev Bhattiprolu on Mon, 30 Jul 2007 23:43:11 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:

| Oleg Nesterov wrote:
| >On 07/26, Pavel Emelyanov wrote:
| >>This is a fix for Sukadev's patch that moved the alloc_pid() call from
| >>do_fork() into copy_process().
| >
| >... and this patch changes almost every line from Sukadev's patch.
|
| It does. My bad :(I have reviewed Suka's patch badly and was sure it
| puts the alloc_pid() right where we need this.

I should have reviewed Pavel's closely too. Sorry.

|
| >Sorry gents, but isn't it better to ask Andrew to drop that patch
| >(which is quite useless by itself), and send a new one which incorporates
| >all necessary changes? Imho, it would be much easier to understand.
|
| Hm... Maybe it's better to ask him to fold these patches together?

I think so, but even dropping my patch is fine with me.

|
| >>@@ -1406,7 +1422,13 @@ long do_fork(unsigned long clone_flags,
| >> if (!IS_ERR(p)) {
| >> struct completion vfork;
| >>
| >>- nr = pid_nr(task_pid(p));
| >>+ /*
| >>+   * this is enough to call pid_nr_ns here, but this if
| >>+   * improves optimisation of regular fork()
| >>+   */
| >>+ nr = (clone_flags & CLONE_NEWPID) ?
| >>+   task_pid_nr_ns(p, current->nsproxy->pid_ns) :
| >>+   task_pid_vnr(p);
| >
| >Shouldn't we do the same for CLONE_PARENT_SETTID in copy_process() ?
| >Otherwise *parent_tidptr may have a wrong value which doesn't match
| >to what fork() returns.
|
| Oops. We should. Thanks :)
|
| >Oleg.
|
| Thanks,
| Pavel

Subject: Re: [PATCH 3/15] kern_siginfo helper
Posted by Sukadev Bhattiprolu on Tue, 31 Jul 2007 00:21:38 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| Oleg Nesterov wrote:
| >On 07/26, Pavel Emelyanov wrote:
| >>TODO: This is more an exploratory patch and modifies only
| >>interfaces
| >> necessary to implement correct signal semantics in pid namespaces.
| >>
| >> If the approach is feasible, we could consistently use 'kern_siginfo'
| >> in other signal interfaces and possibly in 'struct sigqueue'.
| >>
| >> We could modify dequeue_signal() to directly work with 'kern_siginfo'
| >> and remove dequeue_signal_kern_info().
| >
| >Well... I know, it is very easy to blame somebody else's patch, and
| >probably
| >my taste is not good...
| >
| >But honestly, I personally think this approach is a horror, and any
| >alternative
| >is better :)

Hmm. My reasoning was that since siginfo_t was directly "shared" with
user space, extending it even to add a flag was pain.

| >
| >I'd rather change dequeue_signal() so that it takes "struct sigqueue *"
| >parameter instead of "siginfo_t *", or add a new "int *flags".

My earlier version to Containers@ passed in "int *signal_cinit" couple
of levels down and used that in get_signal_to_deliver() but that
looked ugly :-) Passing in sigqueue to dequeue_signal() may be better,
but anyway...

| >
| >OK, this doesn't work anyway, we should do something different. Perhaps
| >just do all checks on sender's side.
|
| Yes. Signal handling in namespaces turned out to be the most complicated
| part of the set. I start thinking to drop this part till we have the "core"
| in -mm tree. Suka, what do you think?

Yes. Lets focus on the core for now and allow privileged user in a child-ns
to terminate the container-init. I will try the signal approach from sender
side also.

|
| >It is a bit strange that this patch is 3/15, and the rest bits in 11/15,
| >not very convenient for the review.
|
| Well, I thought that a split like
| 1. patches for kernel to prepare it for the set
| 2. the set itself
| is the best to review. Maybe I was wrong, but how to make this then?
| E.g. I have a MS_KERNOUNT patch, but its changes are used *much* later.
|
| >Oleg.
| >
| >
|
| Thanks,
| Pavel

Subject: Re: [PATCH 9/15] Move alloc_pid() after the namespace is cloned
Posted by Sukadev Bhattiprolu on Tue, 31 Jul 2007 05:49:11 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| When we create new namespace we will need to allocate the struct pid,
| that will have one extra struct upid in array, comparing to the parent.
| Thus we need to know the new namespace (if any) in alloc_pid() to init
| this struct upid properly, so move the alloc_pid() call lower in
| copy_process().
|
| This is a fix for Sukadev's patch that moved the alloc_pid() call from
| do_fork() into copy_process().
|
| Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
|
| ---
|
| fork.c | 59 ++----------
| 1 files changed, 38 insertions(+), 21 deletions(-)
|
| diff -upr linux-2.6.23-rc1-mm1.orig/kernel/fork.c
| linux-2.6.23-rc1-mm1-7/kernel/fork.c
| --- linux-2.6.23-rc1-mm1.orig/kernel/fork.c 2007-07-26
| 16:34:45.000000000 +0400
| +++ linux-2.6.23-rc1-mm1-7/kernel/fork.c 2007-07-26
| 16:36:37.000000000 +0400
| @@ -1028,16 +1029,9 @@ static struct task_struct *copy_process(
| if (p->binfmt && !try_module_get(p->binfmt->module))
| goto bad_fork_cleanup_put_domain;

```
|
| - if (pid != &init_struct_pid) {
| -  pid = alloc_pid();
| -  if (!pid)
| -   goto bad_fork_put_binfmt_module;
| - }
| -
|  p->did_exec = 0;
|  delayacct_tsk_init(p); /* Must remain after dup_task_struct() */
|  copy_flags(clone_flags, p);
| - p->pid = pid_nr(pid);
|  INIT_LIST_HEAD(&p->children);
|  INIT_LIST_HEAD(&p->sibling);
|  p->vfork_done = NULL;
| @@ -1112,10 +1106,6 @@ static struct task_struct *copy_process(
|  p->blocked_on = NULL; /* not blocked yet */
| #endif
|
| - p->tgid = p->pid;
| - if (clone_flags & CLONE_THREAD)
| -  p->tgid = current->tgid;
| -
|  if ((retval = security_task_alloc(p)))
|   goto bad_fork_cleanup_policy;
|  if ((retval = audit_alloc(p)))
| @@ -1141,6 +1131,17 @@ static struct task_struct *copy_process(
|  if (retval)
|   goto bad_fork_cleanup_namespaces;
|
| + if (pid != &init_struct_pid) {
| +  pid = alloc_pid(task_active_pid_ns(p));
| +  if (!pid)
| +   goto bad_fork_cleanup_namespaces;
```

We should set retval to -EAGAIN before this goto. Otherwise, when we take this goto, retval would be 0 (bc copy_thread() succeeded) and we crash in the caller.

```
| + }
| +
| + p->pid = pid_nr(pid);
| + p->tgid = p->pid;
| + if (clone_flags & CLONE_THREAD)
| +  p->tgid = current->tgid;
| +
|  p->set_child_tid = (clone_flags & CLONE_CHILD_SETTID) ? child_tidptr
|  : NULL;
```

```
|  /*
|   * Clear TID on mm_release()?
| @@ -1237,7 +1242,7 @@ static struct task_struct *copy_process(
|   spin_unlock(&current->sighand->siglock);
|   write_unlock_irq(&tasklist_lock);
|   retval = -ERESTARTNOINTR;
| -  goto bad_fork_cleanup_namespaces;
| +  goto bad_fork_free_pid;
|  }
|
|  if (clone_flags & CLONE_THREAD) {
| @@ -1266,11 +1271,22 @@ static struct task_struct *copy_process(
|    __ptrace_link(p, current->parent);
|
|   if (thread_group_leader(p)) {
| -   p->signal->tty = current->signal->tty;
| -   p->signal->pgrp = task_pgrp_nr(current);
| -   set_task_session(p, task_session_nr(current));
| -   attach_pid(p, PIDTYPE_PGID, task_pgrp(current));
| -   attach_pid(p, PIDTYPE_SID, task_session(current));
| +   if (clone_flags & CLONE_NEWPID) {
| +    p->nsproxy->pid_ns->child_reaper = p;
| +    p->signal->tty = NULL;
| +    p->signal->pgrp = p->pid;
| +    set_task_session(p, p->pid);
| +    attach_pid(p, PIDTYPE_PGID, pid);
| +    attach_pid(p, PIDTYPE_SID, pid);
| +   } else {
| +    p->signal->tty = current->signal->tty;
| +    p->signal->pgrp = task_pgrp_nr(current);
| +    set_task_session(p,
| task_session_nr(current));
| +    attach_pid(p, PIDTYPE_PGID,
| +      task_pgrp(current));
| +    attach_pid(p, PIDTYPE_SID,
| +      task_session(current));
| +   }
|
|    list_add_tail_rcu(&p->tasks, &init_task.tasks);
|    __get_cpu_var(process_counts)++;
| @@ -1288,6 +1304,9 @@ static struct task_struct *copy_process(
|  container_post_fork(p);
|  return p;
|
| +bad_fork_free_pid:
| + if (pid != &init_struct_pid)
| +  free_pid(pid);
| bad_fork_cleanup_namespaces:
```

```
|  exit_task_namespaces(p);
| bad_fork_cleanup_keys:
| @@ -1322,9 +1341,6 @@ bad_fork_cleanup_container:
| #endif
|  container_exit(p, container_callbacks_done);
|  delayacct_tsk_free(p);
| - if (pid != &init_struct_pid)
| -  free_pid(pid);
| -bad_fork_put_binfmt_module:
|  if (p->binfmt)
|   module_put(p->binfmt->module);
| bad_fork_cleanup_put_domain:
| @@ -1406,7 +1422,13 @@ long do_fork(unsigned long clone_flags,
|  if (!IS_ERR(p)) {
|   struct completion vfork;
|
| -  nr = pid_nr(task_pid(p));
| +  /*
| +   * this is enough to call pid_nr_ns here, but this if
| +   * improves optimisation of regular fork()
| +   */
| +  nr = (clone_flags & CLONE_NEWPID) ?
| +   task_pid_nr_ns(p, current->nsproxy->pid_ns) :
| +    task_pid_vnr(p);
|
|   if (clone_flags & CLONE_VFORK) {
|    p->vfork_done = &vfork;
```

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Tue, 31 Jul 2007 09:07:20 GMT
View Forum Message <> Reply to Message

On 07/30, sukadev@us.ibm.com wrote:
>
> --- lx26-23-rc1-mm1.orig/kernel/exit.c 2007-07-26 20:08:16.000000000 -0700
> +++ lx26-23-rc1-mm1/kernel/exit.c 2007-07-30 23:10:30.000000000 -0700
> @@ -915,6 +915,7 @@ fastcall NORET_TYPE void do_exit(long co
> {
>   struct task_struct *tsk = current;
>   int group_dead;
> + struct pid_namespace *pid_ns = tsk->nsproxy->pid_ns;
>
>   profile_task_exit(tsk);
>
> @@ -925,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
>   if (unlikely(!tsk->pid))
>   panic("Attempted to kill the idle task!");

```
>   if (unlikely(tsk == task_child_reaper(tsk))) {
> -  if (task_active_pid_ns(tsk) != &init_pid_ns)
> -   task_active_pid_ns(tsk)->child_reaper =
> -     init_pid_ns.child_reaper;
> +  if (pid_ns != &init_pid_ns) {
> +   zap_pid_ns_processes(pid_ns);
> +   pid_ns->child_reaper = init_pid_ns.child_reaper;
> +  }
>    else
>     panic("Attempted to kill init!");
>   }
```

Just to remind you, this is not right when init is multi-threaded,
we should do this only when the last thread exits.

```
> -static long do_wait(pid_t pid, int options, struct siginfo __user *infop,
> +long do_wait(pid_t pid, int options, struct siginfo __user *infop,
>        int __user *stat_addr, struct rusage __user *ru)
```

Small nit, other in-kernel reapers use sys_wait4(), perhaps we can use
it too, in that case we don't need to export do_wait().

```
> +void zap_pid_ns_processes(struct pid_namespace *pid_ns)
> +{
> + int nr;
> + int rc;
> + int options = WEXITED|__WALL;
> +
> + /*
> +  * We know pid == 1 is terminating. Find remaining pid_ts
> +  * in the namespace, signal them and then wait for them
> +  * exit.
> +  */
> + nr = next_pidmap(pid_ns, 1);
> + while (nr > 0) {
> +  kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
> +  nr = next_pidmap(pid_ns, nr);
> + }
```

Without tasklist_lock held this is not reliable.

Oleg.

Subject: Re: [PATCH 15/15] Hooks over the code to show correct values to user
Posted by Oleg Nesterov on Tue, 31 Jul 2007 10:04:20 GMT
View Forum Message <> Reply to Message

On 07/30, Pavel Emelyanov wrote:
>
> Oleg Nesterov wrote:
> >On 07/26, Pavel Emelyanov wrote:
> >>int
> >>kill_proc(pid_t pid, int sig, int priv)
> >>{
> >>- return kill_proc_info(sig, __si_special(priv), pid);
> >>+ int ret;
> >>+
> >>+ rcu_read_lock();
> >>+ ret = kill_pid_info(sig, __si_special(priv), find_pid(pid));
> >>+ rcu_read_unlock();
> >>+ return ret;
> >>}
> >
> >I think this is wrong. kill_proc() should behave the same as
> >kill_proc_info(),
> >so this change is not needed. With this patch they use different namespaces
> >to find the task, this is not consistent.
>
> Actually, callers of this use tsk->pid (global pid) as an argument, so
> find_vpid() might return wrong value.

Yes I see. But still I don't agree on this issue.

kill_proc() is a simple wrapper on top of kill_proc_info(), not good
to break this. And with this patch they use different namespaces to
search the pid. Imho, not consistent.

Probably we can ignore this for now, but suppose we have some out-of-tree
driver which does kill_proc(pid_number), and the application from non-init
namespace does ioctl(SET_PID_NUMBER, getpid()).

And this is why btw I think find_pid/pid_nr should use active namespace, not
init_pid_ns. That driver can save "struct task_struct*" or "struct pid*".

OK, I understand it is a pain to "fix" the in-tree callers of kill_proc()
(say, we can introduce kill_pid_t() or something), so let's forget this.
In fact, we'd better remove kill_proc(), we should avoid using pid_t, the
callers should be converted to use struct pid.

Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Sukadev Bhattiprolu on Wed, 01 Aug 2007 06:16:16 GMT

Oleg Nesterov [oleg@tv-sign.ru] wrote:
| On 07/30, sukadev@us.ibm.com wrote:
| >
| > --- lx26-23-rc1-mm1.orig/kernel/exit.c 2007-07-26 20:08:16.000000000 -0700
| > +++ lx26-23-rc1-mm1/kernel/exit.c 2007-07-30 23:10:30.000000000 -0700
| > @@ -915,6 +915,7 @@ fastcall NORET_TYPE void do_exit(long co
| > {
| > struct task_struct *tsk = current;
| > int group_dead;
| > + struct pid_namespace *pid_ns = tsk->nsproxy->pid_ns;
| >
| > profile_task_exit(tsk);
| >
| > @@ -925,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
| > if (unlikely(!tsk->pid))
| > panic("Attempted to kill the idle task!");
| > if (unlikely(tsk == task_child_reaper(tsk))) {
| > - if (task_active_pid_ns(tsk) != &init_pid_ns)
| > - task_active_pid_ns(tsk)->child_reaper =
| > - init_pid_ns.child_reaper;
| > + if (pid_ns != &init_pid_ns) {
| > + zap_pid_ns_processes(pid_ns);
| > + pid_ns->child_reaper = init_pid_ns.child_reaper;
| > + }
| > else
| > panic("Attempted to kill init!");
| > }
|
| Just to remind you, this is not right when init is multi-threaded,
| we should do this only when the last thread exits.

Sorry, I needed to clarify somethings about the multi-threaded init. I
got the impresssion that you were sending a patch for the existing bug,
and meant to review/clarify in the context of the patch.

Anyways, re: requirements for multi-threaded init:

Our current definition of is_container_init() and task_child_reaper()
refer only to the main-thread of the container-init (since they check
for pid_t == 1)

If the main-thread is exiting and is the last thread in the group,
we want terminate other processes in the pid ns (simple case).

If the main thread is exiting, but is not the last thread in the
group, should we let it exit and let the next thread in the group
the reaper of the pid ns ?

Then we would have the pid ns w/o a container-init (i.e reaper
does not have a pid_t == 1, but probably does not matter).

And, when this last thread is exiting, we want to terminate other
processes in the ns right ?

|
| > -static long do_wait(pid_t pid, int options, struct siginfo __user *infop,
| > +long do_wait(pid_t pid, int options, struct siginfo __user *infop,
| > int __user *stat_addr, struct rusage __user *ru)
|
| Small nit, other in-kernel reapers use sys_wait4(), perhaps we can use
| it too, in that case we don't need to export do_wait().

Ok.

|
| > +void zap_pid_ns_processes(struct pid_namespace *pid_ns)
| > +{
| > + int nr;
| > + int rc;
| > + int options = WEXITED|__WALL;
| > +
| > + /*
| > + * We know pid == 1 is terminating. Find remaining pid_ts
| > + * in the namespace, signal them and then wait for them
| > + * exit.
| > + */
| > + nr = next_pidmap(pid_ns, 1);
| > + while (nr > 0) {
| > + kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
| > + nr = next_pidmap(pid_ns, nr);
| > + }
|
| Without tasklist_lock held this is not reliable.

Ok. BTW, find_ge_pid() also walks the pidmap, but does not seem to hold
the tasklist_lock. Is that bc its only used in /proc ?

|
| Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Dave Hansen on Wed, 01 Aug 2007 16:00:30 GMT
View Forum Message <> Reply to Message

On Tue, 2007-07-31 at 23:16 -0700, sukadev@us.ibm.com wrote:
> Oleg Nesterov [oleg@tv-sign.ru] wrote:
> | On 07/30, sukadev@us.ibm.com wrote:
> | >
> | > --- lx26-23-rc1-mm1.orig/kernel/exit.c 2007-07-26 20:08:16.000000000 -0700
> | > +++ lx26-23-rc1-mm1/kernel/exit.c 2007-07-30 23:10:30.000000000 -0700
> | > @@ -915,6 +915,7 @@ fastcall NORET_TYPE void do_exit(long co
> | > {
> | >   struct task_struct *tsk = current;
> | >   int group_dead;
> | > + struct pid_namespace *pid_ns = tsk->nsproxy->pid_ns;
> | >
> | >   profile_task_exit(tsk);
> | >
> | > @@ -925,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
> | >   if (unlikely(!tsk->pid))
> | >   panic("Attempted to kill the idle task!");
> | >   if (unlikely(tsk == task_child_reaper(tsk))) {
> | > - if (task_active_pid_ns(tsk) != &init_pid_ns)
> | > -   task_active_pid_ns(tsk)->child_reaper =
> | > -   init_pid_ns.child_reaper;
> | > + if (pid_ns != &init_pid_ns) {
> | > +   zap_pid_ns_processes(pid_ns);
> | > +   pid_ns->child_reaper = init_pid_ns.child_reaper;
> | > + }
> | >   else
> | >   panic("Attempted to kill init!");
> | >   }
> |
> | Just to remind you, this is not right when init is multi-threaded,
> | we should do this only when the last thread exits.
>
> Sorry, I needed to clarify somethings about the multi-threaded init. I
> got the impresssion that you were sending a patch for the existing bug,
> and meant to review/clarify in the context of the patch.
>
> Anyways, re: requirements for multi-threaded init:
>
> Our current definition of is_container_init() and task_child_reaper()
> refer only to the main-thread of the container-init (since they check
> for pid_t == 1)

Remember, the "pid" is actually a tgid:

```
asmlinkage long sys_getpid(void)
{
        return current->tgid;
}
```

So, there are multiple tasks with a "pid" == 1 with a multithreaded
init.

> If the main-thread is exiting and is the last thread in the group,
> we want terminate other processes in the pid ns (simple case).
>
> If the main thread is exiting, but is not the last thread in the
> group, should we let it exit and let the next thread in the group
> the reaper of the pid ns ?

Well, what happens with a multithreaded init today?

-- Dave

Subject: Re: [PATCH 11/15] Signal semantics
Posted by serue on Wed, 01 Aug 2007 16:13:35 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> [snip]
>
> >>| Maybe it's worth disabling cross-namespaces ptracing...
> >>
> >>I think so too. Its probably not a serious limitation ?
> >
> >Several people think we will implement 'namespace entering' through a
> >ptrace hack, where maybe the admin ptraces the init in a child pidns,
>
> Why not implement namespace entering w/o any hacks? :)

I did, as a patch on top of the nsproxy container subsystem. The
response was that that is a hack, and ptrace is cleaner :)

So the current options for namespace entering would be:

* using Cedric's bind_ns() functionality, which assigns an
integer global id to a namespace, and allows a process to
enter a namespace by that global id
* using my nsproxy container subsystem patch, which lets
a process enter another namespace using
echo pid > /container/some/cont/directory/tasks
and eventually might allow construction of custom
namespaces, i.e.
mkdir /container/c1/c2
ln -s /container/c1/c1/network /container/c1/c2/network
echo $$ > /container/c1/c2/tasks

* using ptrace to coerce a process in the target namespace
into forking and executing the desired program.

> >makes it fork, and makes the child execute what it wants (i.e. ps -ef).
> >
> >You're talking about killing that functionality?
>
> No. We're talking about disabling the things that are not supposed
> to work at all.

Uh, well in the abstract that sounds like a sound policy...

-serge

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Wed, 01 Aug 2007 19:48:11 GMT
View Forum Message <> Reply to Message

On 07/31, sukadev@us.ibm.com wrote:
>
> Oleg Nesterov [oleg@tv-sign.ru] wrote:
> | >
> | > @@ -925,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
> | > if (unlikely(!tsk->pid))
> | > panic("Attempted to kill the idle task!");
> | > if (unlikely(tsk == task_child_reaper(tsk))) {
> | > - if (task_active_pid_ns(tsk) != &init_pid_ns)
> | > - task_active_pid_ns(tsk)->child_reaper =
> | > - init_pid_ns.child_reaper;
> | > + if (pid_ns != &init_pid_ns) {
> | > + zap_pid_ns_processes(pid_ns);
> | > + pid_ns->child_reaper = init_pid_ns.child_reaper;
> | > + }
> | > else
> | > panic("Attempted to kill init!");
> | > }
> |
> | Just to remind you, this is not right when init is multi-threaded,
> | we should do this only when the last thread exits.
>
> Sorry, I needed to clarify somethings about the multi-threaded init. I
> got the impresssion that you were sending a patch for the existing bug,
> and meant to review/clarify in the context of the patch.

Ah, sorry, I forgot to send the patch to fix the bug in mainline.
Will try to do tomorrow, please feel free to do this if you wish.

> Our current definition of is_container_init() and task_child_reaper()
> refer only to the main-thread of the container-init (since they check
> for pid_t == 1)

Yes.

> If the main-thread is exiting and is the last thread in the group,
> we want terminate other processes in the pid ns (simple case).

Yes.

> If the main thread is exiting, but is not the last thread in the
> group, should we let it exit and let the next thread in the group
> the reaper of the pid ns ?

We can, but why? The main thread's task_struct can't go away until all
sub-threads exit. Its ->nsproxy will be NULL, but this doesn't matter.

> Then we would have the pid ns w/o a container-init (i.e reaper
> does not have a pid_t == 1, but probably does not matter).
>
> And, when this last thread is exiting, we want to terminate other
> processes in the ns right ?

Yes, when this last thread is exiting, the entire process is exiting.

> | > +void zap_pid_ns_processes(struct pid_namespace *pid_ns)
> | > +{
> | > + int nr;
> | > + int rc;
> | > + int options = WEXITED|__WALL;
> | > +
> | > + /*
> | > + * We know pid == 1 is terminating. Find remaining pid_ts
> | > + * in the namespace, signal them and then wait for them
> | > + * exit.
> | > + */
> | > + nr = next_pidmap(pid_ns, 1);
> | > + while (nr > 0) {
> | > + kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
> | > + nr = next_pidmap(pid_ns, nr);
> | > + }
> |
> | Without tasklist_lock held this is not reliable.
>
> Ok. BTW, find_ge_pid() also walks the pidmap, but does not seem to hold
> the tasklist_lock. Is that bc its only used in /proc ?

Yes, but this is something different. With or without tasklist_lock,
find_ge_pid()/next_tgid() is not "reliable" (note that alloc_pid() doesn't
take tasklist), but this doesn't matter for /proc.

We should take tasklist_lock to prevent the new process creation.
We can have the "false positives" (copy_process() in progress, PGID/SID
pids), but this is OK. Note that copy_process() checks signal_pending()
after write_lock_irq(&tasklist_lock), that is why it helps.

Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Wed, 01 Aug 2007 19:51:13 GMT
View Forum Message <> Reply to Message

On 08/01, Dave Hansen wrote:
>
> > If the main thread is exiting, but is not the last thread in the
> > group, should we let it exit and let the next thread in the group
> > the reaper of the pid ns ?
>
> Well, what happens with a multithreaded init today?

As it was already discussed, the current code is buggy, and should be
fixed.

Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Sukadev Bhattiprolu on Thu, 02 Aug 2007 07:29:58 GMT
View Forum Message <> Reply to Message

Oleg Nesterov [oleg@tv-sign.ru] wrote:
| On 07/31, sukadev@us.ibm.com wrote:
| >
| > Oleg Nesterov [oleg@tv-sign.ru] wrote:
| > | >
| > | > @@ -925,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
| > | > if (unlikely(!tsk->pid))
| > | > panic("Attempted to kill the idle task!");
| > | > if (unlikely(tsk == task_child_reaper(tsk))) {
| > | > - if (task_active_pid_ns(tsk) != &init_pid_ns)
| > | > - task_active_pid_ns(tsk)->child_reaper =
| > | > - init_pid_ns.child_reaper;
| > | > + if (pid_ns != &init_pid_ns) {

| > | > +   zap_pid_ns_processes(pid_ns);
| > | > +   pid_ns->child_reaper = init_pid_ns.child_reaper;
| > | > + }
| > | >   else
| > | >   panic("Attempted to kill init!");
| > | >   }
| > |
| > | Just to remind you, this is not right when init is multi-threaded,
| > | we should do this only when the last thread exits.
| >
| > Sorry, I needed to clarify somethings about the multi-threaded init. I
| > got the impresssion that you were sending a patch for the existing bug,
| > and meant to review/clarify in the context of the patch.
|
| Ah, sorry, I forgot to send the patch to fix the bug in mainline.
| Will try to do tomorrow, please feel free to do this if you wish.

I can do that, but am still a bit confused about this multi-threaded
init :-)

|
| > Our current definition of is_container_init() and task_child_reaper()
| > refer only to the main-thread of the container-init (since they check
| > for pid_t == 1)
|
| Yes.

This means that we cannot have a check like "tsk == task_child_reaper(tsk)"
to properly detect the child reaper process right ?

Its basically a very dumb question - How do we detect a container_init()
in the multi-threaded case ? Should we use "task->tgid == 1" ?

IOW to identify if the last thread of a child reaper is exiting, should we
check "task->tgid == 1" and the "group_dead" flag in do_exit() ?

|
| > If the main-thread is exiting and is the last thread in the group,
| > we want terminate other processes in the pid ns (simple case).
|
| Yes.
|
| > If the main thread is exiting, but is not the last thread in the
| > group, should we let it exit and let the next thread in the group
| > the reaper of the pid ns ?
|
| We can, but why? The main thread's task_struct can't go away until all
| sub-threads exit. Its ->nsproxy will be NULL, but this doesn't matter.

After the main thread exits task_child_reaper() would still refer to
the main thread right ? So when one of the other processes in the
namespace calls forget_original_parent(), it would reparent the process
to the main thread - no ? The main thread still has a valid task_struct,
but it has exited and cannot adapt children...

BTW, are there any actual users of multi-threaded init ? Or is this
something that can be considered outside the "core" patchset and
addressed soon, but separately like the signalling-container-init issue ?

|
| > Then we would have the pid ns w/o a container-init (i.e reaper
| > does not have a pid_t == 1, but probably does not matter).
| >
| > And, when this last thread is exiting, we want to terminate other
| > processes in the ns right ?
|
| Yes, when this last thread is exiting, the entire process is exiting.
|
| > | > +void zap_pid_ns_processes(struct pid_namespace *pid_ns)
| > | > +{
| > | > + int nr;
| > | > + int rc;
| > | > + int options = WEXITED|__WALL;
| > | > +
| > | > + /*
| > | > + * We know pid == 1 is terminating. Find remaining pid_ts
| > | > + * in the namespace, signal them and then wait for them
| > | > + * exit.
| > | > + */
| > | > + nr = next_pidmap(pid_ns, 1);
| > | > + while (nr > 0) {
| > | > + kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
| > | > + nr = next_pidmap(pid_ns, nr);
| > | > + }
| > |
| > | Without tasklist_lock held this is not reliable.
| >
| > Ok. BTW, find_ge_pid() also walks the pidmap, but does not seem to hold
| > the tasklist_lock. Is that bc its only used in /proc ?
|
| Yes, but this is something different. With or without tasklist_lock,
| find_ge_pid()/next_tgid() is not "reliable" (note that alloc_pid() doesn't
| take tasklist), but this doesn't matter for /proc.
|
| We should take tasklist_lock to prevent the new process creation.
| We can have the "false positives" (copy_process() in progress, PGID/SID

| pids), but this is OK. Note that copy_process() checks signal_pending()
| after write_lock_irq(&tasklist_lock), that is why it helps.

Ok. Thx.

|
| Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Sukadev Bhattiprolu on Thu, 02 Aug 2007 07:37:00 GMT
View Forum Message <> Reply to Message

Dave Hansen [haveblue@us.ibm.com] wrote:
| On Tue, 2007-07-31 at 23:16 -0700, sukadev@us.ibm.com wrote:
| > Oleg Nesterov [oleg@tv-sign.ru] wrote:
| > | On 07/30, sukadev@us.ibm.com wrote:
| > | >
| > | > --- lx26-23-rc1-mm1.orig/kernel/exit.c 2007-07-26 20:08:16.000000000 -0700
| > | > +++ lx26-23-rc1-mm1/kernel/exit.c 2007-07-30 23:10:30.000000000 -0700
| > | > @@ -915,6 +915,7 @@ fastcall NORET_TYPE void do_exit(long co
| > | > {
| > | > struct task_struct *tsk = current;
| > | > int group_dead;
| > | > + struct pid_namespace *pid_ns = tsk->nsproxy->pid_ns;
| > | >
| > | > profile_task_exit(tsk);
| > | >
| > | > @@ -925,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
| > | > if (unlikely(!tsk->pid))
| > | > panic("Attempted to kill the idle task!");
| > | > if (unlikely(tsk == task_child_reaper(tsk))) {
| > | > - if (task_active_pid_ns(tsk) != &init_pid_ns)
| > | > - task_active_pid_ns(tsk)->child_reaper =
| > | > - init_pid_ns.child_reaper;
| > | > + if (pid_ns != &init_pid_ns) {
| > | > + zap_pid_ns_processes(pid_ns);
| > | > + pid_ns->child_reaper = init_pid_ns.child_reaper;
| > | > + }
| > | > else
| > | > panic("Attempted to kill init!");
| > | > }
| > |
| > | Just to remind you, this is not right when init is multi-threaded,
| > | we should do this only when the last thread exits.
| >
| > Sorry, I needed to clarify somethings about the multi-threaded init. I
| > got the impresssion that you were sending a patch for the existing bug,

| > and meant to review/clarify in the context of the patch.
| >
| > Anyways, re: requirements for multi-threaded init:
| >
| > Our current definition of is_container_init() and task_child_reaper()
| > refer only to the main-thread of the container-init (since they check
| > for pid_t == 1)
|
| Remember, the "pid" is actually a tgid:
|
| asmlinkage long sys_getpid(void)
| {
| return current->tgid;
| }
|
| So, there are multiple tasks with a "pid" == 1 with a multithreaded
| init.

Yes, and so am now wondering if is_container_init(), is_global_init()
and the "tsk == task_child_reaper(tsk)" checks be replaced with with
something that covers other threads in the reaper ?

|
| > If the main-thread is exiting and is the last thread in the group,
| > we want terminate other processes in the pid ns (simple case).
| >
| > If the main thread is exiting, but is not the last thread in the
| > group, should we let it exit and let the next thread in the group
| > the reaper of the pid ns ?
|
| Well, what happens with a multithreaded init today?
|
| -- Dave

Subject: Re: [PATCH 11/15] Signal semantics
Posted by dev on Thu, 02 Aug 2007 08:35:32 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>
>>[snip]
>>
>>
>>>>>| Maybe it's worth disabling cross-namespaces ptracing...
>>>>
>>>>I think so too. Its probably not a serious limitation ?

>>>
>>>Several people think we will implement 'namespace entering' through a
>>>ptrace hack, where maybe the admin ptraces the init in a child pidns,
>>
>>Why not implement namespace entering w/o any hacks? :)
>
>
> I did, as a patch on top of the nsproxy container subsystem. The
> response was that that is a hack, and ptrace is cleaner :)
>
> So the current options for namespace entering would be:
>
> * using Cedric's bind_ns() functionality, which assigns an
>   integer global id to a namespace, and allows a process to
>   enter a namespace by that global id

looks more or less good and what OVZ actually does.
So I would prefer this one.

> * using my nsproxy container subsystem patch, which lets
>   a process enter another namespace using
>   echo pid > /container/some/cont/directory/tasks
>   and eventually might allow construction of custom
>   namespaces, i.e.
>   mkdir /container/c1/c2
>   ln -s /container/c1/c1/network /container/c1/c2/network
>   echo $$ > /container/c1/c2/tasks

Sound ok and logical as well.

> * using ptrace to coerce a process in the target namespace
>   into forking and executing the desired program.

you'll need to change ptrace interface in this case imho...
doesn't sound ok at all... at least for me. So I agree with Pavel.

>>>makes it fork, and makes the child execute what it wants (i.e. ps -ef).
>>>
>>>You're talking about killing that functionality?
>>
>>No. We're talking about disabling the things that are not supposed
>>to work at all.
>
>
> Uh, well in the abstract that sounds like a sound policy...

Pavel simply meant that no one plans to disable functionality in question.

Thanks,
Kirill

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

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by dev on Thu, 02 Aug 2007 08:37:55 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 08/01, Dave Hansen wrote:
>
>>> If the main thread is exiting, but is not the last thread in the
>>> group, should we let it exit and let the next thread in the group
>>> the reaper of the pid ns ?
>>
>>Well, what happens with a multithreaded init today?
>
>
> As it was already discussed, the current code is buggy, and should be
> fixed.

I'm not that sure it MUST be fixed. There are no multi-threaded init's anywhere.
Oleg, does it worth changing without reasons?

Thanks,
Kirill

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

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Thu, 02 Aug 2007 15:39:34 GMT
View Forum Message <> Reply to Message

On 08/02, sukadev@us.ibm.com wrote:
>
> Oleg Nesterov [oleg@tv-sign.ru] wrote:
> | On 07/31, sukadev@us.ibm.com wrote:
> | >

> | > Oleg Nesterov [oleg@tv-sign.ru] wrote:
> | > | >
> | > | > @@ -925,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
> | > | >   if (unlikely(!tsk->pid))
> | > | >   panic("Attempted to kill the idle task!");
> | > | >   if (unlikely(tsk == task_child_reaper(tsk))) {
> | > | > - if (task_active_pid_ns(tsk) != &init_pid_ns)
> | > | > -   task_active_pid_ns(tsk)->child_reaper =
> | > | > -   init_pid_ns.child_reaper;
> | > | > + if (pid_ns != &init_pid_ns) {
> | > | > +   zap_pid_ns_processes(pid_ns);
> | > | > +   pid_ns->child_reaper = init_pid_ns.child_reaper;

OOPS. I didn't notice this before, but this is not right too (regardless
of multi-threaded init problems).

We should not "reset" ->child_reaper here, we may have exiting tasks
which will re-parent their ->children to global init.

No, we are still /sbin/init of this namespace even if we are exiting,
->child_reaper should point to us, at least until zap_pid_ns_processes()
completes.

> | > Our current definition of is_container_init() and task_child_reaper()
> | > refer only to the main-thread of the container-init (since they check
> | > for pid_t == 1)
> |
> | Yes.
>
> This means that we cannot have a check like "tsk == task_child_reaper(tsk)"
> to properly detect the child reaper process right ?

Yes, we should use "tsk->group_leader == task_child_reaper(tsk)"

> Its basically a very dumb question - How do we detect a container_init()
> in the multi-threaded case ?

Good point. I think is_container_init(tsk) needs a fix:

 - pid = task_pid(tsk);
 + pid = task_pid(tsk->group_leader);

Or, perhaps better, change the callers to use tsk->group_leader.

> Should we use "task->tgid == 1" ?

No, no, "task->tgid == 1" means "global" init.

> IOW to identify if the last thread of a child reaper is exiting, should we
> check "task->tgid == 1" and the "group_dead" flag in do_exit() ?

See above, but yes, as I said before I think we should do this under
the "if (group_dead)" check below.

> | > If the main thread is exiting, but is not the last thread in the
> | > group, should we let it exit and let the next thread in the group
> | > the reaper of the pid ns ?
> |
> | We can, but why? The main thread's task_struct can't go away until all
> | sub-threads exit. Its ->nsproxy will be NULL, but this doesn't matter.
>
> After the main thread exits task_child_reaper() would still refer to
> the main thread right ? So when one of the other processes in the
> namespace calls forget_original_parent(), it would reparent the process
> to the main thread - no ? The main thread still has a valid task_struct,
> but it has exited and cannot adapt children...

Yes it can't, and yes, this is somewhat against the rules.

But, afaics, this should work. Because do_wait() from the alive sub-thread
still can reap the child, note that do_wait() iterates over all sub-threads
->children lists. Please note also that do_notify_parent() uses group
signal, so it will wake up some alive sub-thread.

This is wrong for the "normal" process (because when the last thread exits
main_thread->children is lost), but this seems to be OK for the /sbin/init,
exactly because we are doing zap_pid_ns_processes().

Sukadev, may I ask you to add a fat comment about this in your patch?

> BTW, are there any actual users of multi-threaded init ? Or is this
> something that can be considered outside the "core" patchset and
> addressed soon, but separately like the signalling-container-init issue ?

Well, I don't know. Please also see the reply to Kirill's message...

Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Thu, 02 Aug 2007 16:08:51 GMT
View Forum Message <> Reply to Message

On 08/02, Kirill Korotaev wrote:
>
> Oleg Nesterov wrote:

> > On 08/01, Dave Hansen wrote:
> >
> >>> If the main thread is exiting, but is not the last thread in the
> >>> group, should we let it exit and let the next thread in the group
> >>> the reaper of the pid ns ?
> >>
> >>Well, what happens with a multithreaded init today?
> >
> >
> > As it was already discussed, the current code is buggy, and should be
> > fixed.
>
> I'm not that sure it MUST be fixed. There are no multi-threaded init's anywhere.
> Oleg, does it worth changing without reasons?

I don't know. But the kernel already tries to support multi-threaded init's.
Look at de_thread(), it could be simplified a bit (and we don't need tasklist
lock for zap_other_threads()) if we forbid them.

Still. A non-root user does clone(CLONE_PIDNS), then clone(CLONE_THREAD),
and sys_exit() from the main thread, then proceeds with fork()s. Now this
ns has the global init as a child reaper, and admin can't kill entire pid_ns
by killing its init. Worse, (see the reply to Sukadev' message), we should
not reset pid_ns->child_reaper before zap_pid_ns_processes(). In that case
->child_reaper points to the freed task when the last thread exits, this
means the non-root user can crash the kernel.

Or, some embedded system uses multi-threaded init, and the kernel panics
when the main thread exits.

Perhaps this is just a "quality of implementation" question. sys_exit()
from the main thread should be OK, why /sbin/init should be special?

That said, I personally do not think that multi-threaded init is terribly
useful.

Oleg.

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Oleg Nesterov on Thu, 02 Aug 2007 16:18:24 GMT
View Forum Message <> Reply to Message

On 07/26, Pavel Emelyanov wrote:
>
> The reason to release namespaces after reparenting is that when task
> exits it may send a signal to its parent (SIGCHLD), but if the parent
> has already exited its namespaces there will be no way to decide what
> pid to dever to him - parent can be from different namespace.

I almost forgot about this one...

After reading the whole series, I can't understand the above explanation
any longer. The parent can't be from different namespace, either we have
another sub-thread, or we reparent the child to /sbin/init which should
be from the same namespace.

If we are /sbin/init, we kill all tasks from the same namespace before
doing exit_notify().

Could you clarify?

Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Thu, 02 Aug 2007 17:08:20 GMT
View Forum Message <> Reply to Message

On 08/02, Oleg Nesterov wrote:
>
> On 08/02, Kirill Korotaev wrote:
> >
> > Oleg Nesterov wrote:
> > >
> > > As it was already discussed, the current code is buggy, and should be
> > > fixed.
> >
> > I'm not that sure it MUST be fixed. There are no multi-threaded init's anywhere.
> > Oleg, does it worth changing without reasons?
>
> I don't know. But the kernel already tries to support multi-threaded init's.
> Look at de_thread(), it could be simplified a bit (and we don't need tasklist
> lock for zap_other_threads()) if we forbid them.
>
> Still. A non-root user does clone(CLONE_PIDNS), then clone(CLONE_THREAD),
> and sys_exit() from the main thread, then proceeds with fork()s. Now this
> ns has the global init as a child reaper, and admin can't kill entire pid_ns
> by killing its init. Worse, (see the reply to Sukadev' message), we should
> not reset pid_ns->child_reaper before zap_pid_ns_processes(). In that case
> ->child_reaper points to the freed task when the last thread exits, this

> means the non-root user can crash the kernel.
>
> Or, some embedded system uses multi-threaded init, and the kernel panics
> when the main thread exits.
>
> Perhaps this is just a "quality of implementation" question. sys_exit()
> from the main thread should be OK, why /sbin/init should be special?
>
> That said, I personally do not think that multi-threaded init is terribly
> useful.

So I think the patch below makes sense for now. Note that it removes the
games with pid_ns->child_reaper: this doesn't work currently, and this
has to be modified when we actually support pid namespaces anyway.

Oleg.

```
--- t/kernel/exit.c~MTINIT 2007-07-28 16:58:17.000000000 +0400
+++ t/kernel/exit.c 2007-08-02 20:59:59.000000000 +0400
@@ -895,6 +895,14 @@ static void check_stack_usage(void)
 static inline void check_stack_usage(void) {}
 #endif

+static inline void exit_child_reaper(struct task_struct *tsk)
+{
+ if (likely(tsk->group_leader != child_reaper(tsk)))
+  return;
+
+ panic("Attempted to kill init!");
+}
+
 fastcall NORET_TYPE void do_exit(long code)
 {
  struct task_struct *tsk = current;
@@ -908,13 +916,6 @@ fastcall NORET_TYPE void do_exit(long co
   panic("Aiee, killing interrupt handler!");
  if (unlikely(!tsk->pid))
   panic("Attempted to kill the idle task!");
- if (unlikely(tsk == child_reaper(tsk))) {
-  if (tsk->nsproxy->pid_ns != &init_pid_ns)
-   tsk->nsproxy->pid_ns->child_reaper = init_pid_ns.child_reaper;
-  else
-   panic("Attempted to kill init!");
- }
-

  if (unlikely(current->ptrace & PT_TRACE_EXIT)) {
   current->ptrace_message = code;
```

```
@@ -964,6 +965,7 @@ fastcall NORET_TYPE void do_exit(long co
  }
  group_dead = atomic_dec_and_test(&tsk->signal->live);
  if (group_dead) {
+  exit_child_reaper(tsk);
   hrtimer_cancel(&tsk->signal->real_timer);
   exit_itimers(tsk->signal);
  }
```

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

---

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Sukadev Bhattiprolu on Thu, 02 Aug 2007 17:20:33 GMT
View Forum Message <> Reply to Message

Oleg Nesterov [oleg@tv-sign.ru] wrote:
| On 08/02, sukadev@us.ibm.com wrote:
| >
| > Oleg Nesterov [oleg@tv-sign.ru] wrote:
| > | On 07/31, sukadev@us.ibm.com wrote:
| > | >
| > | > Oleg Nesterov [oleg@tv-sign.ru] wrote:
| > | > | >
| > | > | > @@ -925,9 +926,10 @@ fastcall NORET_TYPE void do_exit(long co
| > | > | >   if (unlikely(!tsk->pid))
| > | > | >   panic("Attempted to kill the idle task!");
| > | > | >   if (unlikely(tsk == task_child_reaper(tsk))) {
| > | > | > - if (task_active_pid_ns(tsk) != &init_pid_ns)
| > | > | > -   task_active_pid_ns(tsk)->child_reaper =
| > | > | > -   init_pid_ns.child_reaper;
| > | > | > + if (pid_ns != &init_pid_ns) {
| > | > | > +   zap_pid_ns_processes(pid_ns);
| > | > | > +   pid_ns->child_reaper = init_pid_ns.child_reaper;
| |
| OOPS. I didn't notice this before, but this is not right too (regardless
| of multi-threaded init problems).
| |
| We should not "reset" ->child_reaper here, we may have exiting tasks
| which will re-parent their ->children to global init.
| |
| No, we are still /sbin/init of this namespace even if we are exiting,
| ->child_reaper should point to us, at least until zap_pid_ns_processes()
| completes.

Yes, we are resetting the reaper _after_ zap_pid_ns_processes() completes
right ? (all other processes in the namespace must have exited).

|
| > | > Our current definition of is_container_init() and task_child_reaper()
| > | > refer only to the main-thread of the container-init (since they check
| > | > for pid_t == 1)
| > |
| > | Yes.
| >
| > This means that we cannot have a check like "tsk == task_child_reaper(tsk)"
| > to properly detect the child reaper process right ?
|
| Yes, we should use "tsk->group_leader == task_child_reaper(tsk)"

Ok.

|
| > Its basically a very dumb question - How do we detect a container_init()
| > in the multi-threaded case ?
|
| Good point. I think is_container_init(tsk) needs a fix:
|
| - pid = task_pid(tsk);
| + pid = task_pid(tsk->group_leader);
|

Ok.

| Or, perhaps better, change the callers to use tsk->group_leader.

Ok.
|
| > Should we use "task->tgid == 1" ?
|
| No, no, "task->tgid == 1" means "global" init.

Grr. I got that mixed up bw my implm and Pavel's :-) task->pid and
task->tgid referred to "active-pid-ns pid" in mine.

|
| > IOW to identify if the last thread of a child reaper is exiting, should we
| > check "task->tgid == 1" and the "group_dead" flag in do_exit() ?
|
| See above, but yes, as I said before I think we should do this under
| the "if (group_dead)" check below.
|

| > | > If the main thread is exiting, but is not the last thread in the
| > | > group, should we let it exit and let the next thread in the group
| > | > the reaper of the pid ns ?
| > |
| > | We can, but why? The main thread's task_struct can't go away until all
| > | sub-threads exit. Its ->nsproxy will be NULL, but this doesn't matter.
| >
| > After the main thread exits task_child_reaper() would still refer to
| > the main thread right ? So when one of the other processes in the
| > namespace calls forget_original_parent(), it would reparent the process
| > to the main thread - no ? The main thread still has a valid task_struct,
| > but it has exited and cannot adapt children...
|
| Yes it can't, and yes, this is somewhat against the rules.
|
| But, afaics, this should work. Because do_wait() from the alive sub-thread
| still can reap the child, note that do_wait() iterates over all sub-threads
| ->children lists. Please note also that do_notify_parent() uses group
| signal, so it will wake up some alive sub-thread.
|
| This is wrong for the "normal" process (because when the last thread exits
| main_thread->children is lost), but this seems to be OK for the /sbin/init,
| exactly because we are doing zap_pid_ns_processes().
|
| Sukadev, may I ask you to add a fat comment about this in your patch?

Sure.

|
| > BTW, are there any actual users of multi-threaded init ? Or is this
| > something that can be considered outside the "core" patchset and
| > addressed soon, but separately like the signalling-container-init issue ?
|
| Well, I don't know. Please also see the reply to Kirill's message...
|
| Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Thu, 02 Aug 2007 17:29:32 GMT
View Forum Message <> Reply to Message

On 08/02, sukadev@us.ibm.com wrote:
>
> Oleg Nesterov [oleg@tv-sign.ru] wrote:
> | > | > | > + if (pid_ns != &init_pid_ns) {
> | > | > | > + zap_pid_ns_processes(pid_ns);
> | > | > | > + pid_ns->child_reaper = init_pid_ns.child_reaper;

> |
> | OOPS. I didn't notice this before, but this is not right too (regardless
> | of multi-threaded init problems).
> |
> | We should not "reset" ->child_reaper here, we may have exiting tasks
> | which will re-parent their ->children to global init.
> |
> | No, we are still /sbin/init of this namespace even if we are exiting,
> | ->child_reaper should point to us, at least until zap_pid_ns_processes()
> | completes.
>
> Yes, we are resetting the reaper _after_ zap_pid_ns_processes() completes
> right ? (all other processes in the namespace must have exited).

OOPS again :) Can't understand how I managed to misread this code.

This means that we should take care about multi-thread init exit,
otherwise the non-root user can crash the kernel.

>From reply to Kirill's message:

> Still. A non-root user does clone(CLONE_PIDNS), then clone(CLONE_THREAD),
> and sys_exit() from the main thread, then proceeds with fork()s. Now this
> ns has the global init as a child reaper, and admin can't kill entire pid_ns
> by killing its init. Worse, (see the reply to Sukadev' message), we should
> not reset pid_ns->child_reaper before zap_pid_ns_processes(). In that case
> ->child_reaper points to the freed task when the last thread exits, this
> means the non-root user can crash the kernel.

Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Sukadev Bhattiprolu on Thu, 02 Aug 2007 18:36:08 GMT
View Forum Message <> Reply to Message

Oleg Nesterov [oleg@tv-sign.ru] wrote:
| On 08/02, sukadev@us.ibm.com wrote:
| >
| > Oleg Nesterov [oleg@tv-sign.ru] wrote:
| > | > | > | > + if (pid_ns != &init_pid_ns) {
| > | > | > | > +   zap_pid_ns_processes(pid_ns);
| > | > | > | > +   pid_ns->child_reaper = init_pid_ns.child_reaper;
| > |
| > | OOPS. I didn't notice this before, but this is not right too (regardless
| > | of multi-threaded init problems).
| > |
| > | We should not "reset" ->child_reaper here, we may have exiting tasks

| > | which will re-parent their ->children to global init.
| > |
| > | No, we are still /sbin/init of this namespace even if we are exiting,
| > | ->child_reaper should point to us, at least until zap_pid_ns_processes()
| > | completes.
| >
| > Yes, we are resetting the reaper _after_ zap_pid_ns_processes() completes
| > right ? (all other processes in the namespace must have exited).
|
| OOPS again :) Can't understand how I managed to misread this code.
|
| This means that we should take care about multi-thread init exit,
| otherwise the non-root user can crash the kernel.
|
| >From reply to Kirill's message:
|
| > Still. A non-root user does clone(CLONE_PIDNS), then clone(CLONE_THREAD),

Agree we should fix the crash. But we need CAP_SYS_ADMIN to clone
pid or other namespaces - this is enforced in copy_namespaces() and
unshare_nsproxy_namespaces()

Suka

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Thu, 02 Aug 2007 18:49:50 GMT
View Forum Message <> Reply to Message

On 08/02, sukadev@us.ibm.com wrote:
>
> Oleg Nesterov [oleg@tv-sign.ru] wrote:
> |
> | This means that we should take care about multi-thread init exit,
> | otherwise the non-root user can crash the kernel.
> |
> | >From reply to Kirill's message:
> |
> | > Still. A non-root user does clone(CLONE_PIDNS), then clone(CLONE_THREAD),
>
> Agree we should fix the crash. But we need CAP_SYS_ADMIN to clone
> pid or other namespaces - this is enforced in copy_namespaces() and
> unshare_nsproxy_namespaces()

Hmm. sys_unshare(CLONE_PIDNS) doesn't (and shouldn't) work anyway, but
I don't see the CAP_SYS_ADMIN check in copy_process()->copy_namespaces()
path.

Perhaps I just missed it (sorry, I already cleared my mbox, so I can't
look at theses patches), but is it a good idea to require CAP_SYS_ADMIN?
I think it would be nice if a normal user can create containers, no?

Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by serue on Thu, 02 Aug 2007 19:13:35 GMT
View Forum Message <> Reply to Message

Quoting Oleg Nesterov (oleg@tv-sign.ru):
> On 08/02, sukadev@us.ibm.com wrote:
> >
> > Oleg Nesterov [oleg@tv-sign.ru] wrote:
> > |
> > | This means that we should take care about multi-thread init exit,
> > | otherwise the non-root user can crash the kernel.
> > |
> > | >From reply to Kirill's message:
> > |
> > | > Still. A non-root user does clone(CLONE_PIDNS), then clone(CLONE_THREAD),
> >
> > Agree we should fix the crash. But we need CAP_SYS_ADMIN to clone
> > pid or other namespaces - this is enforced in copy_namespaces() and
> > unshare_nsproxy_namespaces()
>
> Hmm. sys_unshare(CLONE_PIDNS) doesn't (and shouldn't) work anyway, but
> I don't see the CAP_SYS_ADMIN check in copy_process()->copy_namespaces()
> path.
>
> Perhaps I just missed it (sorry, I already cleared my mbox, so I can't
> look at theses patches), but is it a good idea to require CAP_SYS_ADMIN?
> I think it would be nice if a normal user can create containers, no?

For pid namespaces I can't think of any reason why CAP_SYS_ADMIN should
be needed, since you can't hide processes that way. Same for uts
namespaces.

However for ipc and mount namespaces I'd want to think about it some
more. Any case I can think of right now where they'd be unsafe, is
unsafe anyway, so maybe it's fine...

-serge

Subject: Re: [PATCH 11/15] Signal semantics

Quoting Kirill Korotaev (dev@sw.ru):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >
> >>[snip]
> >>
> >>
> >>>>| Maybe it's worth disabling cross-namespaces ptracing...
> >>>>
> >>>>I think so too. Its probably not a serious limitation ?
> >>>
> >>>Several people think we will implement 'namespace entering' through a
> >>>ptrace hack, where maybe the admin ptraces the init in a child pidns,
> >>
> >>Why not implement namespace entering w/o any hacks? :)
> >
> >
> > I did, as a patch on top of the nsproxy container subsystem. The
> > response was that that is a hack, and ptrace is cleaner :)
> >
> > So the current options for namespace entering would be:
> >
> > * using Cedric's bind_ns() functionality, which assigns an
> >   integer global id to a namespace, and allows a process to
> >   enter a namespace by that global id
>
> looks more or less good and what OVZ actually does.
> So I would prefer this one.

I think this was Cedric's last post of it:

https://lists.linux-foundation.org/pipermail/containers/2007-June/005665.html

However I'm pretty sure Eric would be soundly against this.

> > * using my nsproxy container subsystem patch, which lets
> >   a process enter another namespace using
> >   echo pid > /container/some/cont/directory/tasks
> >   and eventually might allow construction of custom
> >   namespaces, i.e.
> >   mkdir /container/c1/c2
> >   ln -s /container/c1/c1/network /container/c1/c2/network
> >   echo $$ > /container/c1/c2/tasks
>
> Sound ok and logical as well.

I last posted this here:
http://www.mail-archive.com/devel@openvz.org/msg00295.html
In the ensuing thread, the ptrace-based solution is also discussed.

> > * using ptrace to coerce a process in the target namespace
> > into forking and executing the desired program.
>
> you'll need to change ptrace interface in this case imho...
> doesn't sound ok at all... at least for me. So I agree with Pavel.

Well maybe the problem is that while I think of it as ptrace-based, it's
really only like ptrace in that it hijacks another process for a moment
to clone it, but it likely will end up a completely different system
call.

Eric, just curious, have you worked on this at all since february?

-serge

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Sukadev Bhattiprolu on Fri, 03 Aug 2007 06:22:27 GMT
View Forum Message <> Reply to Message

Here is the modified patch that applies on top of Oleg's patch
to fix the error for threaded-init. Hope the "big-fat-comment" in
exit_child_reaper() (about child-reaper inheriting children even
after exiting) makes sense.

From: Pavel Emelyanov <xemul@openvz.org>
Subject: [PATCH 14/15] Destroy pid namespace on init's death

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

Terminate all processes in a namespace when the reaper of the namespace
is exiting. We do this by walking the pidmap of the namespace and sending
SIGKILL to all processes.

Changelog:
[Oleg Nesterov]: In zap_pid_ns_processes() wait for any child
rather than iterating over all pid_ts in the pidmap. Clear
TIF_SIGPENDING flag for successive wait() calls.

```
 [Oleg Nesterov]: Ensure the logic works even with multi-threaded
  container-init process.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Acked-by: Pavel Emelyanov <xemul@openvz.org>

---

 include/linux/pid.h |    2 ++
 kernel/exit.c       |   27 ++++++++++++++++++++++++++-
 kernel/pid.c        |   41 +++++++++++++++++++++++++++++++++++++++++
 3 files changed, 69 insertions(+), 1 deletion(-)

Index: lx26-23-rc1-mm1/include/linux/pid.h
===================================================================
--- lx26-23-rc1-mm1.orig/include/linux/pid.h 2007-08-02 11:03:39.000000000 -0700
+++ lx26-23-rc1-mm1/include/linux/pid.h 2007-08-02 11:06:47.000000000 -0700
@@ -118,6 +118,8 @@ extern struct pid *find_ge_pid(int nr, s

 extern struct pid *alloc_pid(struct pid_namespace *ns);
 extern void FASTCALL(free_pid(struct pid *pid));
+extern void zap_pid_ns_processes(struct pid_namespace *pid_ns,
+  struct task_struct *tsk);

 /*
  * the helpers to get the pid's id seen from different namespaces
Index: lx26-23-rc1-mm1/kernel/exit.c
===================================================================
--- lx26-23-rc1-mm1.orig/kernel/exit.c 2007-08-02 11:06:36.000000000 -0700
+++ lx26-23-rc1-mm1/kernel/exit.c 2007-08-02 23:06:47.000000000 -0700
@@ -916,7 +916,32 @@ static inline void exit_child_reaper(str
  if (likely(tsk->group_leader != task_child_reaper(tsk)))
   return;

- panic("Attempted to kill init!");
+ if (tsk->nsproxy->pid_ns == &init_pid_ns)
+  panic("Attempted to kill init!");
+
+ /*
+  * @tsk is the last thread in the 'container-init' and is exiting.
+  * Terminate all remaining processes in the namespace and reap them
+  * before exiting @tsk.
+  *
+  * Note that @tsk (last thread of container-init) may not necessarily
+  * be the child-reaper (i.e main thread of container-init) of the
+  * namespace i.e the child_reaper may have already exited.
+  *
```

```
+  * Even after a child_reaper exits, we let it inherit orphaned children,
+  * because, pid_ns->child_reaper remains valid as long as there is
+  * at least one living sub-thread in the container init.
+
+  * This living sub-thread of the container-init will be notified when
+  * a child inherited by the 'child-reaper' exits (do_notify_parent()
+  * uses __group_send_sig_info()). Further, when reaping child processes,
+  * do_wait() iterates over children of all living sub threads.
+
+  * i.e even though 'child_reaper' thread is listed as the parent of the
+  * orphaned children, any living sub-thread in the  container-init can
+  * receive notification of the child exiting and reap the child.
+  */
+ zap_pid_ns_processes(tsk->nsproxy->pid_ns, tsk);
 }

 fastcall NORET_TYPE void do_exit(long code)
Index: lx26-23-rc1-mm1/kernel/pid.c
===================================================================
--- lx26-23-rc1-mm1.orig/kernel/pid.c 2007-08-02 11:03:39.000000000 -0700
+++ lx26-23-rc1-mm1/kernel/pid.c 2007-08-02 23:06:40.000000000 -0700
@@ -29,6 +29,7 @@
 #include <linux/pid_namespace.h>
 #include <linux/init_task.h>
 #include <linux/proc_fs.h>
+#include <linux/syscalls.h>

 #define pid_hashfn(nr, ns) \
  hash_long((unsigned long)nr + (unsigned long)ns, pidhash_shift)
@@ -593,6 +594,46 @@ out:
  return new_ns;
 }

+void zap_pid_ns_processes(struct pid_namespace *pid_ns,
+   struct task_struct *reaper)
+{
+ int nr;
+ int rc;
+ pid_t reaper_pid = pid_nr_ns(task_pid(reaper), pid_ns);
+
+ /*
+  * The last thread in the container-init thread group is terminating.
+  * Find remaining pid_ts in the namespace, signal and wait for them
+  * to exit.
+  *
+  * Note:  This signals each threads in the namespace - even those that
+  *    belong to the same thread group, To avoid this, we would have
+  *    to walk the entire tasklist looking a processes in this
```

```
+  *    namespace, but that could be unnecessarily expensive if the
+  *    pid namespace has just a few processes. Or we need to
+  *    maintain a tasklist for each pid namespace.
+  *
+  */
+ read_lock(&tasklist_lock);
+ nr = next_pidmap(pid_ns, 0);
+ while (nr > 0) {
+  if (reaper_pid != nr)
+   kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
+  nr = next_pidmap(pid_ns, nr);
+ }
+ read_unlock(&tasklist_lock);
+
+ do {
+  clear_thread_flag(TIF_SIGPENDING);
+  rc = sys_wait4(-1, NULL, __WALL, NULL);
+ } while (rc != -ECHILD);
+
+
+ /* Don't need a child reaper for this pid namespace anymore */
+ pid_ns->child_reaper = NULL;
+ return;
+}
+
 static void do_free_pid_ns(struct work_struct *w)
 {
  struct pid_namespace *ns, *parent;
```

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Oleg Nesterov on Fri, 03 Aug 2007 10:55:57 GMT
View Forum Message <> Reply to Message

On 08/02, sukadev@us.ibm.com wrote:
>
> --- lx26-23-rc1-mm1.orig/kernel/exit.c 2007-08-02 11:06:36.000000000 -0700
> +++ lx26-23-rc1-mm1/kernel/exit.c 2007-08-02 23:06:47.000000000 -0700
> @@ -916,7 +916,32 @@ static inline void exit_child_reaper(str
>   if (likely(tsk->group_leader != task_child_reaper(tsk)))
>   return;
>
> - panic("Attempted to kill init!");
> + if (tsk->nsproxy->pid_ns == &init_pid_ns)

> + panic("Attempted to kill init!");
> +
> + /*
> + * @tsk is the last thread in the 'container-init' and is exiting.
> + * Terminate all remaining processes in the namespace and reap them
> + * before exiting @tsk.
> + *
> + * Note that @tsk (last thread of container-init) may not necessarily
> + * be the child-reaper (i.e main thread of container-init) of the
> + * namespace i.e the child_reaper may have already exited.
> +   *
> + * Even after a child_reaper exits, we let it inherit orphaned children,
> + * because, pid_ns->child_reaper remains valid as long as there is
> + * at least one living sub-thread in the container init.
> +
> + * This living sub-thread of the container-init will be notified when
> + * a child inherited by the 'child-reaper' exits (do_notify_parent()
> + * uses __group_send_sig_info()). Further, when reaping child processes,
> + * do_wait() iterates over children of all living sub threads.
> +
> + * i.e even though 'child_reaper' thread is listed as the parent of the
> + * orphaned children, any living sub-thread in the container-init can
> + * receive notification of the child exiting and reap the child.
> + */

Great, thanks.

> + zap_pid_ns_processes(tsk->nsproxy->pid_ns, tsk);
> }
>
> +void zap_pid_ns_processes(struct pid_namespace *pid_ns,
> +   struct task_struct *reaper)
> +{
> + int nr;
> + int rc;
> + pid_t reaper_pid = pid_nr_ns(task_pid(reaper), pid_ns);

I personally dislike these paramaters. reaper == current, and it is /sbin/init
always. Not because zap_pid_ns_processes() is always called with reaper == current,
but because zap_pid_ns_processes() can't work otherwise: we are using do_wait()
and assume that forget_original_parent() will re-parent threads to use.

And we use it just to figure out reaper_pid, it is used to avoid sending
SIGKILL to us,

> + read_lock(&tasklist_lock);
> + nr = next_pidmap(pid_ns, 0);
> + while (nr > 0) {

> + if (reaper_pid != nr)
> +   kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
> + nr = next_pidmap(pid_ns, nr);
> + }
> + read_unlock(&tasklist_lock);

But this doesn't work if we are not ->group_leader (iow, when reaper_pid != 1).
Because in that case we are doing kill_proc_info(SIGKILL, SEND_SIG_PRIV, 1),
which sends the signal to entire thread group, and thus to us (because we are
the last alive thread).

This is harmless (and note that it is possible that current was actually killed
with SIGKILL from the parent namespace), but the code imho looks confusing.

I'd suggest to make zap_pid_ns_processes(void), and start the loop from nr == 1.
Or zap_pid_ns_processes(struct pid_namespace *pid_ns).

Oleg.

Subject: Re: [PATCH 14/15] Destroy pid namespace on init's death
Posted by Sukadev Bhattiprolu on Fri, 03 Aug 2007 21:36:36 GMT
View Forum Message <> Reply to Message

Oleg Nesterov [oleg@tv-sign.ru] wrote:
| This is harmless (and note that it is possible that current was actually killed
| with SIGKILL from the parent namespace), but the code imho looks confusing.
|
| I'd suggest to make zap_pid_ns_processes(void), and start the loop from nr == 1.
| Or zap_pid_ns_processes(struct pid_namespace *pid_ns).
|
| Oleg.

Agree. Here is the modified patch.

From: Pavel Emelyanov <xemul@openvz.org>
Subject: [PATCH 14/15] Destroy pid namespace on init's death

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

Terminate all processes in a namespace when the reaper of the namespace
is exiting. We do this by walking the pidmap of the namespace and sending

SIGKILL to all processes.

Changelog:
 [Oleg Nesterov]: In zap_pid_ns_processes() wait for any child
  rather than iterating over all pid_ts in the pidmap. Clear
  TIF_SIGPENDING flag for successive wait() calls.

 [Oleg Nesterov]: Ensure the logic works even with multi-threaded
  container-init process.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Acked-by: Pavel Emelyanov <xemul@openvz.org>

---

 include/linux/pid.h | 1 +
 kernel/exit.c | 27 ++++++++++++++++++++++++-
 kernel/pid.c | 38 ++++++++++++++++++++++++++++++++++++++
 3 files changed, 65 insertions(+), 1 deletion(-)

Index: lx26-23-rc1-mm1/include/linux/pid.h
===
--- lx26-23-rc1-mm1.orig/include/linux/pid.h 2007-08-02 11:03:39.000000000 -0700
+++ lx26-23-rc1-mm1/include/linux/pid.h 2007-08-03 13:29:40.000000000 -0700
@@ -118,6 +118,7 @@ extern struct pid *find_ge_pid(int nr, s

 extern struct pid *alloc_pid(struct pid_namespace *ns);
 extern void FASTCALL(free_pid(struct pid *pid));
+extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

 /*
  * the helpers to get the pid's id seen from different namespaces
Index: lx26-23-rc1-mm1/kernel/exit.c
===
--- lx26-23-rc1-mm1.orig/kernel/exit.c 2007-08-02 11:06:36.000000000 -0700
+++ lx26-23-rc1-mm1/kernel/exit.c 2007-08-03 13:56:37.000000000 -0700
@@ -916,7 +916,32 @@ static inline void exit_child_reaper(str
  if (likely(tsk->group_leader != task_child_reaper(tsk)))
   return;

- panic("Attempted to kill init!");
+ if (tsk->nsproxy->pid_ns == &init_pid_ns)
+ panic("Attempted to kill init!");
+
+ /*
+ * @tsk is the last thread in the 'container-init' and is exiting.
+ * Terminate all remaining processes in the namespace and reap them
+ * before exiting @tsk.

```
+  *
+  * Note that @tsk (last thread of container-init) may not necessarily
+  * be the child-reaper (i.e main thread of container-init) of the
+  * namespace i.e the child_reaper may have already exited.
+   *
+  * Even after a child_reaper exits, we let it inherit orphaned children,
+  * because, pid_ns->child_reaper remains valid as long as there is
+  * at least one living sub-thread in the container init.
+
+  * This living sub-thread of the container-init will be notified when
+  * a child inherited by the 'child-reaper' exits (do_notify_parent()
+  * uses __group_send_sig_info()). Further, when reaping child processes,
+  * do_wait() iterates over children of all living sub threads.
+
+  * i.e even though 'child_reaper' thread is listed as the parent of the
+  * orphaned children, any living sub-thread in the container-init can
+  * perform the role of the child_reaper.
+  */
+ zap_pid_ns_processes(tsk->nsproxy->pid_ns);
 }

 fastcall NORET_TYPE void do_exit(long code)
Index: lx26-23-rc1-mm1/kernel/pid.c
===================================================================
--- lx26-23-rc1-mm1.orig/kernel/pid.c 2007-08-02 11:03:39.000000000 -0700
+++ lx26-23-rc1-mm1/kernel/pid.c 2007-08-03 13:56:12.000000000 -0700
@@ -29,6 +29,7 @@
 #include <linux/pid_namespace.h>
 #include <linux/init_task.h>
 #include <linux/proc_fs.h>
+#include <linux/syscalls.h>

 #define pid_hashfn(nr, ns) \
  hash_long((unsigned long)nr + (unsigned long)ns, pidhash_shift)
@@ -593,6 +594,43 @@ out:
  return new_ns;
 }

+void zap_pid_ns_processes(struct pid_namespace *pid_ns)
+{
+ int nr;
+ int rc;
+
+ /*
+  * The last thread in the container-init thread group is terminating.
+  * Find remaining pid_ts in the namespace, signal and wait for them
+  * to exit.
+  *
```

```
+  * Note:  This signals each threads in the namespace - even those that
+  *    belong to the same thread group, To avoid this, we would have
+  *    to walk the entire tasklist looking a processes in this
+  *    namespace, but that could be unnecessarily expensive if the
+  *    pid namespace has just a few processes. Or we need to
+  *    maintain a tasklist for each pid namespace.
+  *
+  */
+ read_lock(&tasklist_lock);
+ nr = next_pidmap(pid_ns, 1);
+ while (nr > 0) {
+  kill_proc_info(SIGKILL, SEND_SIG_PRIV, nr);
+  nr = next_pidmap(pid_ns, nr);
+ }
+ read_unlock(&tasklist_lock);
+
+ do {
+  clear_thread_flag(TIF_SIGPENDING);
+  rc = sys_wait4(-1, NULL, __WALL, NULL);
+ } while (rc != -ECHILD);
+
+
+ /* Child reaper for the pid namespace is going away */
+ pid_ns->child_reaper = NULL;
+ return;
+}
+
 static void do_free_pid_ns(struct work_struct *w)
 {
  struct pid_namespace *ns, *parent;
```

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Pavel Emelianov on Mon, 06 Aug 2007 08:00:44 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 07/26, Pavel Emelyanov wrote:
>> The reason to release namespaces after reparenting is that when task
>> exits it may send a signal to its parent (SIGCHLD), but if the parent
>> has already exited its namespaces there will be no way to decide what
>> pid to dever to him - parent can be from different namespace.
>
> I almost forgot about this one...

>
> After reading the whole series, I can't understand the above explanation
> any longer. The parent can't be from different namespace, either we have
> another sub-thread, or we reparent the child to /sbin/init which should
> be from the same namespace.

If the child that is a new namespace's init is exiting its parent is from the
different namespace. Moreover, we will probably want to implement "entering"
the pid namespace, so having tasks with parents from another namespace will
be OK.

> If we are /sbin/init, we kill all tasks from the same namespace before
> doing exit_notify().
>
> Could you clarify?
>
> Oleg.
>
>

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Oleg Nesterov on Mon, 06 Aug 2007 09:54:21 GMT
View Forum Message <> Reply to Message

On 08/06, Pavel Emelyanov wrote:
> Oleg Nesterov wrote:
> >On 07/26, Pavel Emelyanov wrote:
> >>The reason to release namespaces after reparenting is that when task
> >>exits it may send a signal to its parent (SIGCHLD), but if the parent
> >>has already exited its namespaces there will be no way to decide what
> >>pid to dever to him - parent can be from different namespace.
> >
> >I almost forgot about this one...
> >
> >After reading the whole series, I can't understand the above explanation
> >any longer. The parent can't be from different namespace, either we have
> >another sub-thread, or we reparent the child to /sbin/init which should
> >be from the same namespace.
>
> If the child that is a new namespace's init is exiting its parent is from the
> different namespace.

In that case it doesn't have childs. The were SIGKILL'ed before exit_notify().

> Moreover, we will probably want to implement "entering"
> the pid namespace, so having tasks with parents from another namespace will
> be OK.

Well. I saw this word "entering", but I don't know the meaning. Just curious,
could you explain?

And, if an exiting task has a child which is already from another namespace,
why can't we release our namespace before re-parenting? I guess I need to
know what "entering" means to understand this...

Oleg.

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Pavel Emelianov on Mon, 06 Aug 2007 09:58:50 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 08/06, Pavel Emelyanov wrote:
>> Oleg Nesterov wrote:
>>> On 07/26, Pavel Emelyanov wrote:
>>>> The reason to release namespaces after reparenting is that when task
>>>> exits it may send a signal to its parent (SIGCHLD), but if the parent
>>>> has already exited its namespaces there will be no way to decide what
>>>> pid to dever to him - parent can be from different namespace.
>>> I almost forgot about this one...
>>>
>>> After reading the whole series, I can't understand the above explanation
>>> any longer. The parent can't be from different namespace, either we have
>>> another sub-thread, or we reparent the child to /sbin/init which should
>>> be from the same namespace.
>> If the child that is a new namespace's init is exiting its parent is from the
>> different namespace.
>
> In that case it doesn't have childs. The were SIGKILL'ed before exit_notify().

It does not, but it's parent - does :)

>> Moreover, we will probably want to implement "entering"
>> the pid namespace, so having tasks with parents from another namespace will
>> be OK.
>
> Well. I saw this word "entering", but I don't know the meaning. Just curious,
> could you explain?

"Entering" means "moving task to arbitrary namespace"

> And, if an exiting task has a child which is already from another namespace,
> why can't we release our namespace before re-parenting? I guess I need to
> know what "entering" means to understand this...

One of the desired actions was the following:
1. task X clones the new namespace with the child Y as this namespace's init;
2. task X waits for SIGCHILD to come informing that the namespace is dead.
In this scenario we need to set the Y's pid as it is seen from X's
namespace in siginfo.

> Oleg.
>
>

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Oleg Nesterov on Mon, 06 Aug 2007 10:38:31 GMT
View Forum Message <> Reply to Message

On 08/06, Pavel Emelyanov wrote:
>
> Oleg Nesterov wrote:
> >On 08/06, Pavel Emelyanov wrote:
> >>Oleg Nesterov wrote:
> >>>On 07/26, Pavel Emelyanov wrote:
> >>>>The reason to release namespaces after reparenting is that when task
> >>>>exits it may send a signal to its parent (SIGCHLD), but if the parent
> >>>>has already exited its namespaces there will be no way to decide what
> >>>>pid to dever to him - parent can be from different namespace.
> >>>I almost forgot about this one...
> >>>
> >>>After reading the whole series, I can't understand the above explanation
> >>>any longer. The parent can't be from different namespace, either we have
> >>>another sub-thread, or we reparent the child to /sbin/init which should
> >>>be from the same namespace.
> >>If the child that is a new namespace's init is exiting its parent is from
> >>the
> >>different namespace.
> >
> >In that case it doesn't have childs. The were SIGKILL'ed before
> >exit_notify().
>
> It does not, but it's parent - does :)

Yes. But in that case forget_original_parent() is no-op! This means that
it is not needed to move exit_task_namepsace() after.

> >>Moreover, we will probably want to implement "entering"
> >>the pid namespace, so having tasks with parents from another namespace
> >>will
> >>be OK.

> >
> >Well. I saw this word "entering", but I don't know the meaning. Just
> >curious,
> >could you explain?
>
> "Entering" means "moving task to arbitrary namespace"

Heh. Very much nontrivial, good luck :) At least we need to grow ->numbers[],
and if its pid was pinned...

> >And, if an exiting task has a child which is already from another
> >namespace,
> >why can't we release our namespace before re-parenting? I guess I need to
> >know what "entering" means to understand this...
>
> One of the desired actions was the following:
> 1. task X clones the new namespace with the child Y as this namespace's
> init;
> 2. task X waits for SIGCHILD to come informing that the namespace is dead.
> In this scenario we need to set the Y's pid as it is seen from X's
> namespace in siginfo.

Yes sure. But again, how this is connected to "we should do exit_task_namespace()
after forget_original_parent()" ?

I guess I missed something stupid and simple...

Oleg.

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Pavel Emelianov on Mon, 06 Aug 2007 11:21:46 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 08/06, Pavel Emelyanov wrote:
>> Oleg Nesterov wrote:
>>> On 08/06, Pavel Emelyanov wrote:
>>>> Oleg Nesterov wrote:
>>>>> On 07/26, Pavel Emelyanov wrote:
>>>>>> The reason to release namespaces after reparenting is that when task
>>>>>> exits it may send a signal to its parent (SIGCHLD), but if the parent
>>>>>> has already exited its namespaces there will be no way to decide what
>>>>>> pid to dever to him - parent can be from different namespace.
>>>>> I almost forgot about this one...
>>>>>
>>>>> After reading the whole series, I can't understand the above explanation
>>>>> any longer. The parent can't be from different namespace, either we have

>>>>> another sub-thread, or we reparent the child to /sbin/init which should
>>>>> be from the same namespace.
>>>> If the child that is a new namespace's init is exiting its parent is from
>>>> the
>>>> different namespace.
>>> In that case it doesn't have childs. The were SIGKILL'ed before
>>> exit_notify().
>> It does not, but it's parent - does :)
>
> Yes. But in that case forget_original_parent() is no-op! This means that
> it is not needed to move exit_task_namepsace() after.
>
>>>> Moreover, we will probably want to implement "entering"
>>>> the pid namespace, so having tasks with parents from another namespace
>>>> will
>>>> be OK.
>>> Well. I saw this word "entering", but I don't know the meaning. Just
>>> curious,
>>> could you explain?
>> "Entering" means "moving task to arbitrary namespace"
>
> Heh. Very much nontrivial, good luck :) At least we need to grow ->numbers[],
> and if its pid was pinned...
>
>>> And, if an exiting task has a child which is already from another
>>> namespace,
>>> why can't we release our namespace before re-parenting? I guess I need to
>>> know what "entering" means to understand this...
>> One of the desired actions was the following:
>> 1. task X clones the new namespace with the child Y as this namespace's
>> init;
>> 2. task X waits for SIGCHILD to come informing that the namespace is dead.
>> In this scenario we need to set the Y's pid as it is seen from X's
>> namespace in siginfo.
>
> Yes sure. But again, how this is connected to "we should do exit_task_namespace()
> after forget_original_parent()" ?
>
> I guess I missed something stupid and simple...

If task X is exiting and has already exit_task_namespaces()-ed task
Y will OOPs during its exit in determining parent's namespace. I agree
that in that case this is not important what namespace X belongs to,
but we need to handle the race with changing the nsproxy from not-NULL
to NULL. This is OK to make this under task_lock() but what to add
extra locking for if we can avoid it?

> Oleg.

>
>

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Pavel Emelianov on Mon, 06 Aug 2007 11:29:18 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 08/06, Pavel Emelyanov wrote:
>> Oleg Nesterov wrote:
>>> On 08/06, Pavel Emelyanov wrote:
>>>> Oleg Nesterov wrote:
>>>>> On 07/26, Pavel Emelyanov wrote:
>>>>>> The reason to release namespaces after reparenting is that when task
>>>>>> exits it may send a signal to its parent (SIGCHLD), but if the parent
>>>>>> has already exited its namespaces there will be no way to decide what
>>>>>> pid to dever to him - parent can be from different namespace.
>>>>> I almost forgot about this one...
>>>>>
>>>>> After reading the whole series, I can't understand the above explanation
>>>>> any longer. The parent can't be from different namespace, either we have
>>>>> another sub-thread, or we reparent the child to /sbin/init which should
>>>>> be from the same namespace.
>>>> If the child that is a new namespace's init is exiting its parent is from
>>>> the
>>>> different namespace.
>>> In that case it doesn't have childs. The were SIGKILL'ed before
>>> exit_notify().
>> It does not, but it's parent - does :)
>
> Yes. But in that case forget_original_parent() is no-op! This means that
> it is not needed to move exit_task_namepsace() after.
>
>>>> Moreover, we will probably want to implement "entering"
>>>> the pid namespace, so having tasks with parents from another namespace
>>>> will
>>>> be OK.
>>> Well. I saw this word "entering", but I don't know the meaning. Just
>>> curious,
>>> could you explain?
>> "Entering" means "moving task to arbitrary namespace"
>
> Heh. Very much nontrivial, good luck :) At least we need to grow ->numbers[],
> and if its pid was pinned...
>
>>> And, if an exiting task has a child which is already from another
>>> namespace,

>>> why can't we release our namespace before re-parenting? I guess I need to
>>> know what "entering" means to understand this...
>> One of the desired actions was the following:
>> 1. task X clones the new namespace with the child Y as this namespace's
>> init;
>> 2. task X waits for SIGCHILD to come informing that the namespace is dead.
>> In this scenario we need to set the Y's pid as it is seen from X's
>> namespace in siginfo.
>
> Yes sure. But again, how this is connected to "we should do exit_task_namespace()
> after forget_original_parent()" ?
>
> I guess I missed something stupid and simple...

In other words. Let task X live in init_pid_ns, task Y is his child and lives
int another namespace. task X and task Y both die. This will happen:

1. Task X call exit_task_namespaces()
   and sets its nsproxy to NULL
                                1. Task Y is going to notify the
                                   parent (X) and dereferences its
                                   nsproxy -> OOPS
2. Task X reparents all its children

If we move the exit_task_namespace this will happen:

1. Task X reparents all its children
2. Task X call exit_task_namespaces()
   and sets its nsproxy to NULL

In such case is tasy Y will dereference the parent's nsproxy it will not
OOPS because either its parent will be not X already, or X's nsproxy is
not yet released.

> Oleg.
>
>

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Oleg Nesterov on Mon, 06 Aug 2007 12:48:20 GMT
View Forum Message <> Reply to Message

On 08/06, Pavel Emelyanov wrote:
>
> Oleg Nesterov wrote:
> >On 08/06, Pavel Emelyanov wrote:
> >>Oleg Nesterov wrote:

> >>>On 08/06, Pavel Emelyanov wrote:
> >>>>Oleg Nesterov wrote:
> >>>>>On 07/26, Pavel Emelyanov wrote:
> >>>>>>The reason to release namespaces after reparenting is that when task
> >>>>>>exits it may send a signal to its parent (SIGCHLD), but if the parent
> >>>>>>has already exited its namespaces there will be no way to decide what
> >>>>>>pid to dever to him - parent can be from different namespace.
> >>>>>I almost forgot about this one...
> >>>>>
> >>>I guess I missed something stupid and simple...
>
> In other words. Let task X live in init_pid_ns, task Y is his child and
> lives
> int another namespace. task X and task Y both die. This will happen:
>
> 1. Task X call exit_task_namespaces()
> and sets its nsproxy to NULL

Ah, got it, thanks. So the problem is not namespace itself (parent's or child's), there are still valid (even if different but related).

We just can't get ->parent->nsproxy. I was greatly confused by the "parent can be from different namespace" above. We have exactly same problem if namespaces are not differ.

IOW, the problem is: we can't clear ->nsproxy (exit_task_namespaces) until we get rid of ->children. This have nothing to do with different namespace.

Oleg.

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Oleg Nesterov on Mon, 06 Aug 2007 12:52:08 GMT
View Forum Message <> Reply to Message

On 08/06, Pavel Emelyanov wrote:
>
> If task X is exiting and has already exit_task_namespaces()-ed task
> Y will OOPs during its exit in determining parent's namespace. I agree
> that in that case this is not important what namespace X belongs to,
> but we need to handle the race with changing the nsproxy from not-NULL
> to NULL. This is OK to make this under task_lock() but what to add
> extra locking for if we can avoid it?

No, we can't take task_lock() under write_lock(tasklist).

Oleg.

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Pavel Emelianov on Mon, 06 Aug 2007 13:36:51 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 08/06, Pavel Emelyanov wrote:
>> Oleg Nesterov wrote:
>>> On 08/06, Pavel Emelyanov wrote:
>>>> Oleg Nesterov wrote:
>>>>> On 08/06, Pavel Emelyanov wrote:
>>>>>> Oleg Nesterov wrote:
>>>>>>> On 07/26, Pavel Emelyanov wrote:
>>>>>>>> The reason to release namespaces after reparenting is that when task
>>>>>>>> exits it may send a signal to its parent (SIGCHLD), but if the parent
>>>>>>>> has already exited its namespaces there will be no way to decide what
>>>>>>>> pid to dever to him - parent can be from different namespace.
>>>>>>> I almost forgot about this one...
>>>>>>>
>>>>> I guess I missed something stupid and simple...
>> In other words. Let task X live in init_pid_ns, task Y is his child and
>> lives
>> int another namespace. task X and task Y both die. This will happen:
>>
>> 1. Task X call exit_task_namespaces()
>> and sets its nsproxy to NULL
>
> Ah, got it, thanks. So the problem is not namespace itself (parent's or
> child's), there are still valid (even if different but related).
>
> We just can't get ->parent->nsproxy. I was greatly confused by the "parent
> can be from different namespace" above. We have exactly same problem if
> namespaces are not differ.
>
> IOW, the problem is: we can't clear ->nsproxy (exit_task_namespaces) until
> we get rid of ->children. This have nothing to do with different namespace.

No. If the parent is always in the same namespace we do not need to
get its nsproxy :) Problem is exactly in that the parent's namespace
is to be known.

> Oleg.
>
>

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Pavel Emelianov on Mon, 06 Aug 2007 13:38:03 GMT
View Forum Message <> Reply to Message

Oleg Nesterov wrote:
> On 08/06, Pavel Emelyanov wrote:
>> If task X is exiting and has already exit_task_namespaces()-ed task
>> Y will OOPs during its exit in determining parent's namespace. I agree
>> that in that case this is not important what namespace X belongs to,
>> but we need to handle the race with changing the nsproxy from not-NULL
>> to NULL. This is OK to make this under task_lock() but what to add
>> extra locking for if we can avoid it?
>
> No, we can't take task_lock() under write_lock(tasklist).

I meant that we could save the namspace earlier (w/o tasklist),
carry it up to the place we need and release it later. But all
this is too complex and it's easier to get rid of children and
then release the namespaces.

> Oleg.
>
>

Subject: Re: [PATCH 1/15] Move exit_task_namespaces()
Posted by Oleg Nesterov on Mon, 06 Aug 2007 13:55:53 GMT
View Forum Message <> Reply to Message

On 08/06, Pavel Emelyanov wrote:
>
> Oleg Nesterov wrote:
> >On 08/06, Pavel Emelyanov wrote:
> >>Oleg Nesterov wrote:
> >>>On 08/06, Pavel Emelyanov wrote:
> >>>>Oleg Nesterov wrote:
> >>>>>On 07/26, Pavel Emelyanov wrote:
> >>>>>The reason to release namespaces after reparenting is that when task
> >>>>>exits it may send a signal to its parent (SIGCHLD), but if the
> >>>>>parent
> >>>>>has already exited its namespaces there will be no way to decide
> >>>>>what
> >>>>>pid to dever to him - parent can be from different namespace.
> >>>>I almost forgot about this one...
> >>>>
> >>>I guess I missed something stupid and simple...
> >>In other words. Let task X live in init_pid_ns, task Y is his child and
> >>lives
> >>int another namespace. task X and task Y both die. This will happen:
> >>

> >>1. Task X call exit_task_namespaces()
> >> and sets its nsproxy to NULL
> >
> >Ah, got it, thanks. So the problem is not namespace itself (parent's or
> >child's), there are still valid (even if different but related).
> >
> >We just can't get ->parent->nsproxy. I was greatly confused by the "parent
> >can be from different namespace" above. We have exactly same problem if
> >namespaces are not differ.
> >
> >IOW, the problem is: we can't clear ->nsproxy (exit_task_namespaces) until
> >we get rid of ->children. This have nothing to do with different namespace.
>
> No. If the parent is always in the same namespace we do not need to
> get its nsproxy :) Problem is exactly in that the parent's namespace
> is to be known.

Yes yes, I see. I meant: once do_notify_parent() was modified to use
parent->nsproxy to figure out correct pid_t, that problem has nothing
to do with namespaces, it is just parent->nsproxy access.

But this is not safe, btw? do_notify_parent() can get parent->nsproxy
which is under destruction (sys_unshare). Then we read its ->pid_ns,
but at this time "struct nsproxy" could be kmem_cache_free()'ed ?

Of course, this is just theoretical, irqs are disabled, and the window
is tiny.

Oleg.
