Subject: [RFC][PATCH 0/16] Enable cloning of pid namespace
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:03:03 GMT
View Forum Message <> Reply to Message

Dave, Serge,

Here is my current pid namespace patchset. There are still a couple of
issues I am investigating, but appreciate any feedback.

Suka

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [RFC][PATCH 0/16] Enable clone pid namespace

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.

Imagine it like a directory structure that has several processes
chroot'd into it:

```
/
|-- dir0
|   |-- subdir0
|   `-- subdir1
`-- dir1
    |-- subdir2
    `-- subdir3
```

A process in the '/' dir can see all of the directories. However, one

chrooted into dir0 can only see subdirs 0 and 1. It can not see dir1,
or subdirs 2 and 3. Pid namespaces work in the same way.

Following are the patches in this set. They apply to the 2.6.21-mm2 kernel.

[PATCH 1/16] Define/use task_active_pid_ns() wrapper

[PATCH 2/16] Rename pid_nr function

[PATCH 3/16] Rename child_reaper function.

[PATCH 4/16] Use pid_to_nr() in process info functions

[PATCH 5/16] Use task_pid() to find leader's pid

[PATCH 6/16] Pid namespaces: define is_global_init()

[PATCH 7/16] Move alloc_pid call to copy_process

[PATCH 8/16] Define and use pid->upid_list list.

[PATCH 9/16] Use pid ns from upid_list list

[PATCH 10/16] Define CLONE_NEWPID flag

[PATCH 11/16] Enable cloning pid namespace

[PATCH 12/16] Terminate processes in a ns when reaper is exiting.

[PATCH 13/16] Remove proc_mnt's use for killing inodes

[PATCH 14/16] Introduce proc_mnt for pid_ns

[PATCH 15/16] Enable signalling child reaper from parent ns.

[PATCH 16/16] Move inline functions to sched.h

Changelog:

(Summary of major changes since last post to Containers@).
These also listed in changelog of relevant patches.

2.6.21-mm2-pidns3:

- Rename some functions for clarity: child_reaper() to
  task_child_reaper(), pid_nr() to pid_to_nr(), task_pid_ns() to
  task_active_pid_ns().

- Define/use helpers to access parent process info: task_parent_tgid(),
  task_parent_pid(), task_tracer_pid() task_tracer_tgid().

    - 'struct upid' used to be called 'struct pid_nr' and a list of these
  were hanging off of 'struct pid'. So, renamed 'struct pid_nr' and
  now hold them in a statically sized array in 'struct pid' since the
  number of 'struct upid's for a process is known at process-creation
  time.

- Allow terminating the child reaper (and hence the entire namespace)
  from an ancestor namespace.

Dictionary of some process data structures/functions.

struct pid:
 Kernel layer process id with lightweight reference to process.

struct pid_link:
 Connects the task_struct of a process to its struct pids

pid_t:
 User space layer integer process id

struct pid_namespace:
 Process id namespace

struct upid:
 Connects a pid_t with the pid_ns in which it is valid. Each
 'struct pid' has a list of these (pid->upid_list) representing
 every pid_t by which it is known in user space.

pid_to_nr():
 Function call (used to be pid_nr()) - return the pid_t of a
 process depending on the pid namespace of caller.

pidmap:
 A bitmap representing which pid_t values are in use/free.
 Used to efficiently assign a pid_t to newly created processes.

TODO:
 As of 2.6.21-mm2-pidns3:

 1. There is a memory/ref-count leak of struct pids (not all struct pids
    are getting freed). Problem seems to be with the patch that
    introduces proc mnt for a pid-ns.

 2. Fix fcntl() and get_signal_to_deliver() to ensure a process cannot
    get signals directly from a process in a descendant namespace.

3. proc_get_sb()/proc_fill_super() should get a reference to the pid
   ns and hold the reference till user space unmounts /proc.

4. Include pid_namespace in pid_hash() so processes with same pid_t
   in different namespaces are on different hash lists.

5. Allow any legitimiate signals that a reaper can get from within its
   namespace. (we currently block all signals to reaper from within
   the namespace).

6. Sending SIGKILL to the child reaper of a namespace terminates the
   namespace. But if the namespace remounted /proc from userspace,
   /proc would remain mounted even after reaper and other process in
   the namespace go away.

7. Identify clone flags that should not be specified with CLONE_NEWPID
   and return -EINVAL from copy_process(), if they are specified.

8. Add privilege check for CLONE_NEWPID.

9. Do we need a lock in task_child_reaper() for any race with do_exit()

   10. Consider maintaining a per-pid namespace tasklist. Use that list
       to terminate processes in the namespace more efficiently. Such a
       tasklist may also be useful to freeze or checkpoint an application.

To test/create pidnamespaces, save the C program attached below as
pidns_exec.c and run following commands:

$ make pidns_exec

Test1: Create a shell in a new pid namespace and ensure only processes
       in that namespace are visible from the namespace.

```
$ cat lxc-wrap.sh
#!/bin/sh

if [ $$ -eq 1 ]; then
 echo "$$ (`basename $0`): Executing in nested pid ns..."
fi

port_num=$1;
if [ $# -lt 1 ]; then
    port_num=2100
fi

mount -t proc lxcproc /proc
```

/usr/sbin/sshd -D -p $port_num
umount /proc

$ ./pidns_exec ./lxc-wrap.sh

Now from another window, login to the
system where above sshd is running

$ ssh -p 2100 <user@hostname>

$ sleep 1000 &

$ ps -e -o"pid,ppid,pgid,sid,cmd"

The pids shown should start at 1. Other processes
in init_pid_ns should not be visible.

Test2: Ensure processes in child pid namespace are visible from
init_pid_ns.

From a third window, ssh into the test system on default port
22 (i.e we should now be in init-pid-ns) and run:

$ ps -e -o"pid,ppid,pgid,sid,cmd"

All processes in the system, including the 'sleep 1000' process
above should be visible.

Note the different pids for the "sleep 1000" in Test1 and Test2.

Additional tests:

Run a kernel build and/or LTP from the pid namespace created
in Test1. (We ran kernel build. LTP TBD on current patchset).

Finally: reboot the system and ensure no crashes during shutdown :-)

```
/* begin pidns_exec.c */

/*
 * This code is trimmed-down version of Serge Hallyn's ns_exec.c
 * and focusses on testing cloning of pid namespace.
 */

/*
 * To create nested namespace:
```

```
 *
 *  pidns_exec -- lxc-wrap.sh
 *   In a pid namespace that is 1 level below init_pid_ns,
 *   execute lxc-wrap.sh
 *
 *  pidns_exec -N 20 -- lxc-wrap.sh
 *   In a pid namespace that is 20 levels below init_pid_ns,
 *   execute lxc-wrap.sh
 *
 * To run as fork bomb:
 *
 *  pidns_exec -F 32768 -- /bin/ls -l
 *
 *   Create 32768 processes that execute /bin/ls -l in
 *   current directory.
 *
 * Note that '--' is needed to separate args to pidns_exec from args to
 * command.
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <libgen.h>
#include <errno.h>
#include <sys/wait.h>

#ifndef SYS_unshare
#ifdef __NR_unshare
#define SYS_unshare __NR_unshare
#elif __i386__
#define SYS_unshare 310
#elif __ia64__
#define SYS_unshare 1296
#elif __x86_64__
#define SYS_unshare 272
#elif __s390x__
#define SYS_unshare 303
#elif __powerpc__
#define SYS_unshare 282
#else
#error "unshare not supported on this architecure."
#endif
#endif

#define CLONE_FS        0x00000200 /* set if fs info shared between processes */
#define CLONE_NEWNS     0x00020000      /* New namespace group? */
#define CLONE_NEWPID    0x10000000      /* New pid namespace */
```

```
static const char* procname;

enum test_mode {
 NESTED_NS,
 FORK_BOMB
};

static void usage(const char *name)
{
 printf("Usage:\n", name);
 printf("       %s [-N <num>] [command [arg ..]]\n", name);
 printf("       %s [-F <num>] [command [arg ..]]\n", name);
 printf("       %s [-h]\n", name);
 printf("\n\n");
 printf("  -F <num> (F)ork bomb <num> processes and have them run "
    "'command'\n\n");
 printf("  -N <num> Run 'command' in a pid ns (N)ested <num> "
    "levels below init_pid_ns\n\n");
 printf("  -h  this message\n");
 printf("\n");
 printf("If no options are specified, assume '-N 1'\n\n");
 printf("(C) Copyright IBM Corp. 2006\n");
 printf("\n");
 exit(1);
}

int do_exec(char *vargv[])
{
 char **argv = (char **)vargv;

 execve(argv[0], argv, __environ);
 perror("execve");
 exit(9);
}

do_fork_bomb(int num_procs, char *argv[])
{
 int i;
 int status;

 for (i = 0; i < num_procs; i++) {
  if (fork() == 0) {
   printf("%d: Fork-bomb: Ppid %d, Prog %s\n",
     getpid(), getppid(), (char *)argv[0]);
   do_exec(argv);
  }
 }
```

```
 while(wait(&status) != -1)
  ;

 _exit(0);
}

struct child_arg {
 char **argv;
 int  cur_depth;
 int  max_depth;
};

unshare_mnt_ns(struct child_arg *carg)
{
 /* To remount /proc in new pid ns, unshare mount ns */

 if (syscall(SYS_unshare, CLONE_NEWNS|CLONE_FS) < 0) {
  perror("unshare");
  exit(1);
 }
}

void *prepare_for_clone()
{
 void *stack;
 void *childstack;

 stack = malloc(getpagesize());
 if (!stack) {
  perror("malloc");
  exit(1);
 }
 childstack = stack + getpagesize();
 return childstack;
}

int do_nested_pid_ns(struct child_arg *carg)
{
 void *childstack;
 int ret;
 unsigned long flags = CLONE_NEWPID;
 int status;
 int pid;
 int rc;

 if (carg->cur_depth++ == carg->max_depth) {
  do_exec(carg->argv);
```

```
  return;
 }

 unshare_mnt_ns(carg);

 childstack = prepare_for_clone();

 ret = clone(do_nested_pid_ns, childstack, flags, (void *)carg);
 if (ret == -1) {
  perror("clone");
  return -1;
 }

 rc = waitpid(-1, &status, __WALL);
}

int main(int argc, char *argv[])
{
 int c;
 int status;
 struct child_arg carg;
 int rc;
 int mode = NESTED_NS;
 int num_procs = 32768;

 procname = basename(argv[0]);

 carg.max_depth = 1;
 carg.cur_depth = 0;

 while ((c = getopt(argc, argv, "F:hN:")) != EOF) {
  switch (c) {
  case 'F':
   mode = FORK_BOMB;
   num_procs = atoi(argv[optind-1]);
   break;
  case 'N':
   mode = NESTED_NS;
   carg.max_depth = atoi(argv[optind-1]);
   break;
  case 'h':
  default:
   usage(procname);
  }
 };

 carg.argv = &argv[optind];
```

```
 if (mode == NESTED_NS) {
  printf("%d (%s): Creating nested pid ns of depth %d\n",
    getpid(), procname, carg.max_depth);
  do_nested_pid_ns(&carg);
 } else {
  do_fork_bomb(num_procs, carg.argv);
 }

 while(waitpid(-1, &status, __WALL) != -1)
  ;

 printf("%d (%s): Exiting\n", getpid(), procname);
 exit(0);
}
```

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

---

Subject: [RFC][PATCH 01/16] Define/use task_active_pid_ns() wrapper
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:08:20 GMT
View Forum Message <> Reply to Message

Subject: Define/use task_active_pid_ns() wrapper

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

A process is visible in one or more pid namespaces, as it is known by some pid_t in every ancestor pid namespace. Every time the process forks, the child process also gets a pid_t in every ancestor pid namespace.

While a process is visible in >=1 pid namespaces, it can see pid_t's in only one pid namespace. We call this pid namespace it's "active pid namespace", and it is always the youngest pid namespace in which the process is known.

This patch defines/uses a wrapper to find the active pid namespace of a process. The implementation of the wrapper will be changed in subsequent patches when support for multiple pid namespaces are added.

Changelog:
 2.6.21-rc6-mm1:
 - Rename task_pid_ns() to task_active_pid_ns() to reflect that
   a process can have multiple pid namespaces.

```
Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 fs/exec.c                  |    2 +-
 fs/proc/proc_misc.c        |    3 ++-
 include/linux/pid_namespace.h |    7 ++++++-
 kernel/exit.c              |    5 +++--
 kernel/pid.c               |    4 ++--
 5 files changed, 14 insertions(+), 7 deletions(-)

Index: lx26-21-mm2/include/linux/pid_namespace.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 17:49:52.000000000 -0700
@@ -37,9 +37,14 @@ static inline void put_pid_ns(struct pid
  kref_put(&ns->kref, free_pid_ns);
 }

+static inline struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
+{
+ return tsk->nsproxy->pid_ns;
+}
+
 static inline struct task_struct *child_reaper(struct task_struct *tsk)
 {
- return init_pid_ns.child_reaper;
+ return task_active_pid_ns(tsk)->child_reaper;
 }

 #endif /* _LINUX_PID_NS_H */
Index: lx26-21-mm2/fs/exec.c
===================================================================
--- lx26-21-mm2.orig/fs/exec.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/fs/exec.c 2007-05-22 17:49:52.000000000 -0700
@@ -629,7 +629,7 @@ static int de_thread(struct task_struct
   * so it is safe to do it under read_lock.
   */
  if (unlikely(tsk->group_leader == child_reaper(tsk)))
-  tsk->nsproxy->pid_ns->child_reaper = tsk;
+  task_active_pid_ns(tsk)->child_reaper = tsk;

  zap_other_threads(tsk);
  read_unlock(&tasklist_lock);
Index: lx26-21-mm2/fs/proc/proc_misc.c
===================================================================
--- lx26-21-mm2.orig/fs/proc/proc_misc.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/fs/proc/proc_misc.c 2007-05-22 16:59:34.000000000 -0700
@@ -94,7 +94,8 @@ static int loadavg_read_proc(char *page,
   LOAD_INT(a), LOAD_FRAC(a),
```

```
   LOAD_INT(b), LOAD_FRAC(b),
   LOAD_INT(c), LOAD_FRAC(c),
-  nr_running(), nr_threads, current->nsproxy->pid_ns->last_pid);
+  nr_running(), nr_threads,
+  task_active_pid_ns(current)->last_pid);
  return proc_calc_metrics(page, start, off, count, eof, len);
 }

Index: lx26-21-mm2/kernel/exit.c
===================================================================
--- lx26-21-mm2.orig/kernel/exit.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/kernel/exit.c 2007-05-22 17:49:52.000000000 -0700
@@ -876,8 +876,9 @@ fastcall NORET_TYPE void do_exit(long co
  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;
+  if (task_active_pid_ns(tsk) != &init_pid_ns)
+   task_active_pid_ns(tsk)->child_reaper =
+     init_pid_ns.child_reaper;
   else
    panic("Attempted to kill init!");
  }
Index: lx26-21-mm2/kernel/pid.c
===================================================================
--- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/kernel/pid.c 2007-05-22 17:49:52.000000000 -0700
@@ -213,7 +213,7 @@ struct pid *alloc_pid(void)
  if (!pid)
   goto out;

- nr = alloc_pidmap(current->nsproxy->pid_ns);
+ nr = alloc_pidmap(task_active_pid_ns(current));
  if (nr < 0)
   goto out_free;

@@ -358,7 +358,7 @@ struct pid *find_ge_pid(int nr)
   pid = find_pid(nr);
   if (pid)
    break;
-  nr = next_pidmap(current->nsproxy->pid_ns, nr);
+  nr = next_pidmap(task_active_pid_ns(current), nr);
  } while (nr > 0);

  return pid;
```

Subject: [RFC][PATCH 02/16] Rename pid_nr function
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:08:52 GMT
View Forum Message <> Reply to Message

Subject: Rename pid_nr function

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

Rename pid_nr() function to pid_to_nr() which is more descriptive
and will hopefully cause less confusion with new structure/functions
being added to support multiple pid namespaces.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
drivers/char/n_r3964.c | 7 ++++---
drivers/char/tty_io.c | 4 ++--
fs/autofs/root.c | 2 +-
fs/fcntl.c | 2 +-
fs/proc/array.c | 2 +-
fs/smbfs/proc.c | 2 +-
include/linux/pid.h | 2 +-
ipc/mqueue.c | 2 +-
kernel/fork.c | 2 +-
kernel/sysctl.c | 2 +-
10 files changed, 14 insertions(+), 13 deletions(-)

Index: lx26-21-mm2/include/linux/pid.h
==========
--- lx26-21-mm2.orig/include/linux/pid.h 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/include/linux/pid.h 2007-05-22 16:59:40.000000000 -0700
@@ -98,7 +98,7 @@ extern struct pid *find_ge_pid(int nr);
extern struct pid *alloc_pid(void);
extern void FASTCALL(free_pid(struct pid *pid));

-static inline pid_t pid_nr(struct pid *pid)
+static inline pid_t pid_to_nr(struct pid *pid)
{
pid_t nr = 0;
if (pid)
Index: lx26-21-mm2/drivers/char/n_r3964.c
==========
--- lx26-21-mm2.orig/drivers/char/n_r3964.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/drivers/char/n_r3964.c 2007-05-22 16:59:40.000000000 -0700
@@ -771,7 +771,7 @@ static int enable_signals(struct r3964_i

```
    if (pClient->pid == pid) {
     TRACE_PS("removing client %d from client list",
-      pid_nr(pid));
+      pid_to_nr(pid));
     *ppClient = pClient->next;
     while (pClient->msg_count) {
      pMsg = remove_msg(pInfo, pClient);
@@ -801,7 +801,8 @@ static int enable_signals(struct r3964_i
    if (pClient == NULL)
     return -ENOMEM;

-   TRACE_PS("add client %d to client list", pid_nr(pid));
+   TRACE_PS("add client %d to client list",
+     pid_to_nr(pid));
    spin_lock_init(&pClient->lock);
    pClient->sig_flags = arg;
    pClient->pid = get_pid(pid);
@@ -933,7 +934,7 @@ static void remove_client_block(struct r
 {
  struct r3964_block_header *block;

- TRACE_PS("remove_client_block PID %d", pid_nr(pClient->pid));
+ TRACE_PS("remove_client_block PID %d", pid_to_nr(pClient->pid));

  block = pClient->next_block_to_read;
  if (block) {
Index: lx26-21-mm2/drivers/char/tty_io.c
===================================================================
--- lx26-21-mm2.orig/drivers/char/tty_io.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/drivers/char/tty_io.c 2007-05-22 16:59:40.000000000 -0700
@@ -3048,7 +3048,7 @@ 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_to_nr(real_tty->pgrp), p);
 }

 /**
@@ -3119,7 +3119,7 @@ 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_to_nr(real_tty->session), p);
 }
```

```
 /**
Index: lx26-21-mm2/fs/autofs/root.c
===================================================================
--- lx26-21-mm2.orig/fs/autofs/root.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/fs/autofs/root.c 2007-05-22 16:59:40.000000000 -0700
@@ -214,7 +214,7 @@ static struct dentry *autofs_root_lookup

  oz_mode = autofs_oz_mode(sbi);
  DPRINTK(("autofs_lookup: pid = %u, pgrp = %u, catatonic = %d, "
-    "oz_mode = %d\n", pid_nr(task_pid(current)),
+    "oz_mode = %d\n", pid_to_nr(task_pid(current)),
     process_group(current), sbi->catatonic,
     oz_mode));

Index: lx26-21-mm2/fs/fcntl.c
===================================================================
--- lx26-21-mm2.orig/fs/fcntl.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/fs/fcntl.c 2007-05-22 16:59:40.000000000 -0700
@@ -305,7 +305,7 @@ 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_to_nr(filp->f_owner.pid);
  if (filp->f_owner.pid_type == PIDTYPE_PGID)
   pid = -pid;
  read_unlock(&filp->f_owner.lock);
Index: lx26-21-mm2/fs/proc/array.c
===================================================================
--- lx26-21-mm2.orig/fs/proc/array.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/fs/proc/array.c 2007-05-22 16:59:40.000000000 -0700
@@ -351,7 +351,7 @@ 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_to_nr(sig->tty->pgrp);
    tty_nr = new_encode_dev(tty_devnum(sig->tty));
   }

Index: lx26-21-mm2/fs/smbfs/proc.c
===================================================================
--- lx26-21-mm2.orig/fs/smbfs/proc.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/fs/smbfs/proc.c 2007-05-22 16:59:41.000000000 -0700
@@ -972,7 +972,7 @@ smb_newconn(struct smb_sb_info *server,

  VERBOSE("protocol=%d, max_xmit=%d, pid=%d capabilities=0x%x\n",
   server->opt.protocol, server->opt.max_xmit,
```

```
-  pid_nr(server->conn_pid), server->opt.capabilities);
+  pid_to_nr(server->conn_pid), server->opt.capabilities);

  /* FIXME: this really should be done by smbmount. */
  if (server->opt.max_xmit > SMB_MAX_PACKET_SIZE) {
Index: lx26-21-mm2/ipc/mqueue.c
===================================================================
--- lx26-21-mm2.orig/ipc/mqueue.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/ipc/mqueue.c 2007-05-22 16:59:41.000000000 -0700
@@ -337,7 +337,7 @@ 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_to_nr(info->notify_owner));
  spin_unlock(&info->lock);
  buffer[sizeof(buffer)-1] = '\0';
  slen = strlen(buffer)+1;
Index: lx26-21-mm2/kernel/fork.c
===================================================================
--- lx26-21-mm2.orig/kernel/fork.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/kernel/fork.c 2007-05-22 16:59:41.000000000 -0700
@@ -1028,7 +1028,7 @@ static struct task_struct *copy_process(
  p->did_exec = 0;
  delayacct_tsk_init(p); /* Must remain after dup_task_struct() */
  copy_flags(clone_flags, p);
- p->pid = pid_nr(pid);
+ p->pid = pid_to_nr(pid);

  INIT_LIST_HEAD(&p->children);
  INIT_LIST_HEAD(&p->sibling);
Index: lx26-21-mm2/kernel/sysctl.c
===================================================================
--- lx26-21-mm2.orig/kernel/sysctl.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/kernel/sysctl.c 2007-05-22 16:59:41.000000000 -0700
@@ -2119,7 +2119,7 @@ static int proc_do_cad_pid(ctl_table *ta
  pid_t tmp;
  int r;

- tmp = pid_nr(cad_pid);
+ tmp = pid_to_nr(cad_pid);

  r = __do_proc_dointvec(&tmp, table, write, filp, buffer,
          lenp, ppos, NULL, NULL);
```

Subject: [RFC][PATCH 03/16] Rename child_reaper function
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:09:28 GMT
View Forum Message <> Reply to Message

Subject: Rename child_reaper function.

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

Rename the child_reaper() function to task_child_reaper() to be
inline with other task_* functions and to distinguish the function
from struct pid_namspace.child_reaper.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 fs/exec.c | 2 +-
 include/linux/pid_namespace.h | 2 +-
 kernel/exit.c | 4 ++--
 kernel/signal.c | 2 +-
 4 files changed, 5 insertions(+), 5 deletions(-)

Index: lx26-21-mm2/fs/exec.c
===
--- lx26-21-mm2.orig/fs/exec.c 2007-05-22 16:59:34.000000000 -0700
+++ lx26-21-mm2/fs/exec.c 2007-05-22 16:59:42.000000000 -0700
@@ -628,7 +628,7 @@ static int de_thread(struct task_struct
   * Reparenting needs write_lock on tasklist_lock,
   * so it is safe to do it under read_lock.
   */
- if (unlikely(tsk->group_leader == child_reaper(tsk)))
+ if (unlikely(tsk->group_leader == task_child_reaper(tsk)))
   task_active_pid_ns(tsk)->child_reaper = tsk;

  zap_other_threads(tsk);
Index: lx26-21-mm2/kernel/exit.c
===
--- lx26-21-mm2.orig/kernel/exit.c 2007-05-22 16:59:34.000000000 -0700
+++ lx26-21-mm2/kernel/exit.c 2007-05-22 16:59:42.000000000 -0700
@@ -687,7 +687,7 @@ forget_original_parent(struct task_struc
  do {
   reaper = next_thread(reaper);
   if (reaper == father) {
-   reaper = child_reaper(father);
+   reaper = task_child_reaper(father);
    break;
   }
  } while (reaper->exit_state);
@@ -875,7 +875,7 @@ 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 (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;
Index: lx26-21-mm2/kernel/signal.c
===================================================================
--- lx26-21-mm2.orig/kernel/signal.c 2007-05-22 16:59:29.000000000 -0700
+++ lx26-21-mm2/kernel/signal.c 2007-05-22 16:59:42.000000000 -0700
@@ -1912,7 +1912,7 @@ relock:
    * within that pid space. It can of course get signals from
    * its parent pid space.
    */
-  if (current == child_reaper(current))
+  if (current == task_child_reaper(current))
    continue;

   if (sig_kernel_stop(signr)) {
Index: lx26-21-mm2/include/linux/pid_namespace.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:34.000000000 -0700
+++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 16:59:42.000000000 -0700
@@ -42,7 +42,7 @@ static inline struct pid_namespace *task
  return tsk->nsproxy->pid_ns;
 }

-static inline struct task_struct *child_reaper(struct task_struct *tsk)
+static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
 {
  return task_active_pid_ns(tsk)->child_reaper;
 }
```

_____

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

---

Subject: [RFC][PATCH 04/16] Use pid_to_nr() in process info functions
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:09:57 GMT
View Forum Message <> Reply to Message

Subject: Use pid_to_nr() in process info functions

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

Use pid_to_nr() function in getppid(), getpgid() and getsid() functions

so they return the correct pid_t for processes in multiple pid namespaces.

Note: We don't need pid_to_nr() in getpid() because the process always
    "sees itself" as being in its active pid namespace. Using pid_to_nr()
    in getpid() would unnecessarily hurt its performance.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 include/linux/sched.h | 24 ++++++++++++++++++++++
 kernel/sys.c | 10 +++++-----
 kernel/timer.c | 2 +-
 3 files changed, 30 insertions(+), 6 deletions(-)

Index: lx26-21-mm2/include/linux/sched.h
==========
--- lx26-21-mm2.orig/include/linux/sched.h 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/include/linux/sched.h 2007-05-22 16:59:44.000000000 -0700
@@ -1124,6 +1124,30 @@ static inline struct pid *task_tgid(stru
  return task->group_leader->pids[PIDTYPE_PID].pid;
 }

+/* NOTE: Caller must hold rcu_readlock() */
+static inline struct pid *task_parent_tgid(struct task_struct *task)
+{
+ return task_tgid(rcu_dereference(task->real_parent));
+}
+
+/* NOTE: Caller must hold rcu_readlock() */
+static inline struct pid *task_parent_pid(struct task_struct *task)
+{
+ return task_pid(rcu_dereference(task->real_parent));
+}
+
+/* NOTE: Caller must hold rcu_readlock() */
+static inline struct pid *task_tracer_tgid(struct task_struct *task)
+{
+ return task_tgid(rcu_dereference(task->parent));
+}
+
+/* NOTE: Caller must hold rcu_readlock() */
+static inline struct pid *task_tracer_pid(struct task_struct *task)
+{
+ return task_pid(rcu_dereference(task->parent));
+}
+
 static inline struct pid *task_pgrp(struct task_struct *task)
 {
  return task->group_leader->pids[PIDTYPE_PGID].pid;

```
Index: lx26-21-mm2/kernel/timer.c
===================================================================
--- lx26-21-mm2.orig/kernel/timer.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/kernel/timer.c 2007-05-22 16:59:44.000000000 -0700
@@ -945,7 +945,7 @@ asmlinkage long sys_getppid(void)
  int pid;

  rcu_read_lock();
- pid = rcu_dereference(current->real_parent)->tgid;
+ pid = pid_to_nr(task_parent_tgid(current));
  rcu_read_unlock();

  return pid;
Index: lx26-21-mm2/kernel/sys.c
===================================================================
--- lx26-21-mm2.orig/kernel/sys.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/kernel/sys.c 2007-05-22 16:59:44.000000000 -0700
@@ -1500,7 +1500,7 @@ out:
 asmlinkage long sys_getpgid(pid_t pid)
 {
  if (!pid)
-  return process_group(current);
+  return pid_to_nr(task_pgrp(current));
  else {
   int retval;
   struct task_struct *p;
@@ -1512,7 +1512,7 @@ asmlinkage long sys_getpgid(pid_t pid)
    if (p) {
     retval = security_task_getpgid(p);
     if (!retval)
-     retval = process_group(p);
+     retval = pid_to_nr(task_pgrp(p));
    }
    read_unlock(&tasklist_lock);
    return retval;
@@ -1524,7 +1524,7 @@ asmlinkage long sys_getpgid(pid_t pid)
 asmlinkage long sys_getpgrp(void)
 {
  /* SMP - assuming writes are word atomic this is fine */
- return process_group(current);
+ return pid_to_nr(task_pgrp(current));
 }

 #endif
@@ -1532,7 +1532,7 @@ asmlinkage long sys_getpgrp(void)
 asmlinkage long sys_getsid(pid_t pid)
 {
  if (!pid)
```

```
-  return process_session(current);
+  return pid_to_nr(task_session(current));
  else {
   int retval;
   struct task_struct *p;
@@ -1544,7 +1544,7 @@ asmlinkage long sys_getsid(pid_t pid)
   if (p) {
    retval = security_task_getsid(p);
    if (!retval)
-    retval = process_session(p);
+    retval = pid_to_nr(task_session(p));
   }
   read_unlock(&tasklist_lock);
   return retval;
```

Subject: [RFC][PATCH 05/16] Use task_pid() to find leader's pid
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:10:32 GMT
View Forum Message <> Reply to Message

Subject: Use task_pid() to find leader's pid

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

Use task_pid() to get leader's pid since find_pid() cannot be used
after detach_pid(). See comments in the code below for more details.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 fs/exec.c | 9 ++++++++-
 1 file changed, 8 insertions(+), 1 deletion(-)

Index: lx26-21-mm2/fs/exec.c
```
===================================================================
--- lx26-21-mm2.orig/fs/exec.c 2007-05-22 16:59:42.000000000 -0700
+++ lx26-21-mm2/fs/exec.c 2007-05-22 16:59:45.000000000 -0700
@@ -707,10 +707,17 @@ static int de_thread(struct task_struct
    * The old leader becomes a thread of the this thread group.
    * Note: The old leader also uses this pid until release_task
    *       is called.  Odd but simple and correct.
+   * Note: With multiple pid namespaces, active pid namespace of
+   *   a process is stored in its struct pid. The detach_pid
+   *   below frees the struct pid, so we will have no notion
+   *   of an active pid namespace until we complete the
```

```
+   *   subsequent attach_pid(). Which means - calls like
+   *   find_pid()/pid_to_nr() return NULL and cannot be used
+   *   between the detach_pid() and attach_pid() calls.
    */
   detach_pid(tsk, PIDTYPE_PID);
   tsk->pid = leader->pid;
-  attach_pid(tsk, PIDTYPE_PID,  find_pid(tsk->pid));
+  attach_pid(tsk, PIDTYPE_PID,  task_pid(leader));
   transfer_pid(leader, tsk, PIDTYPE_PGID);
   transfer_pid(leader, tsk, PIDTYPE_SID);
   list_replace_rcu(&leader->tasks, &tsk->tasks);
```

Subject: [RFC][PATCH 06/16] Define is_global_init()
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:11:00 GMT
View Forum Message <> Reply to Message

Subject: Define is_global_init()

From: Serge E. Hallyn <serue@us.ibm.com>

is_init() is an ambiguous name for the pid==1 check. Split it into
is_global_init() and is_container_init().

A container init has it's tsk->pid == 1.

A global init also has it's tsk->pid == 1, and it's active pid namespace
is the init_pid_ns.

Changelog:

 2.6.21-mm2-pidns2:

 - [Sukadev Bhattiprolu] Changed is_container_init() calls in {powerpc,
   ppc,avr32}/traps.c for the _exception() call to is_global_init().
   This way, we kill only the container if the container's init has a
   bug rather than force a kernel panic.

Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 arch/alpha/mm/fault.c | 2 +-
 arch/arm/mm/fault.c | 2 +-

```
 arch/arm26/mm/fault.c                |    2 +-
 arch/avr32/kernel/traps.c            |    2 +-
 arch/avr32/mm/fault.c                |    6 +++---
 arch/i386/lib/usercopy.c             |    2 +-
 arch/i386/mm/fault.c                 |    2 +-
 arch/ia64/mm/fault.c                 |    2 +-
 arch/m32r/mm/fault.c                 |    2 +-
 arch/m68k/mm/fault.c                 |    2 +-
 arch/mips/mm/fault.c                 |    2 +-
 arch/powerpc/kernel/traps.c          |    2 +-
 arch/powerpc/mm/fault.c              |    2 +-
 arch/powerpc/platforms/pseries/ras.c |    2 +-
 arch/ppc/kernel/traps.c              |    2 +-
 arch/ppc/mm/fault.c                  |    2 +-
 arch/s390/lib/uaccess_pt.c           |    2 +-
 arch/s390/mm/fault.c                 |    2 +-
 arch/sh/mm/fault.c                   |    2 +-
 arch/sh64/mm/fault.c                 |    6 +++---
 arch/um/kernel/trap.c                |    2 +-
 arch/x86_64/mm/fault.c               |    4 ++--
 arch/xtensa/mm/fault.c               |    2 +-
 drivers/char/sysrq.c                 |    2 +-
 include/linux/sched.h                |   13 +++++----------
 kernel/capability.c                  |    3 ++-
 kernel/exit.c                        |    2 +-
 kernel/kexec.c                       |    2 +-
 kernel/pid.c                         |   21 +++++++++++++++++++++
 kernel/sysctl.c                      |    2 +-
 mm/oom_kill.c                        |    4 ++--
 security/commoncap.c                 |    2 +-
 32 files changed, 61 insertions(+), 46 deletions(-)

Index: lx26-21-mm2/arch/alpha/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/alpha/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/alpha/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -192,7 +192,7 @@ do_page_fault(unsigned long address, uns
  /* We ran out of memory, or some other thing happened to us that
     made us unable to handle the page fault gracefully.  */
  out_of_memory:
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/arm/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/arm/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
```

```
+++ lx26-21-mm2/arch/arm/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -197,7 +197,7 @@ survive:
   return fault;
  }

- if (!is_init(tsk))
+ if (!is_global_init(tsk))
   goto out;

  /*
Index: lx26-21-mm2/arch/arm26/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/arm26/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/arm26/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -185,7 +185,7 @@ survive:
  }

  fault = -3; /* out of memory */
- if (!is_init(tsk))
+ if (!is_global_init(tsk))
   goto out;

  /*
Index: lx26-21-mm2/arch/i386/lib/usercopy.c
===================================================================
--- lx26-21-mm2.orig/arch/i386/lib/usercopy.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/i386/lib/usercopy.c 2007-05-22 16:59:46.000000000 -0700
@@ -748,7 +748,7 @@ survive:
    retval = get_user_pages(current, current->mm,
      (unsigned long )to, 1, 1, 0, &pg, NULL);

-   if (retval == -ENOMEM && is_init(current)) {
+   if (retval == -ENOMEM && is_global_init(current)) {
     up_read(&current->mm->mmap_sem);
     congestion_wait(WRITE, HZ/50);
     goto survive;
Index: lx26-21-mm2/arch/i386/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/i386/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/i386/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -576,7 +576,7 @@ no_context:
  */
 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(tsk)) {
+ if (is_global_init(tsk)) {
   yield();
   down_read(&mm->mmap_sem);
```

```
   goto survive;
Index: lx26-21-mm2/arch/ia64/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/ia64/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/ia64/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -279,7 +279,7 @@ ia64_do_page_fault (unsigned long addres

   out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/m32r/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/m32r/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/m32r/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -272,7 +272,7 @@ no_context:
  */
 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(tsk)) {
+ if (is_global_init(tsk)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/m68k/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/m68k/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/m68k/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -181,7 +181,7 @@ good_area:
  */
 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/mips/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/mips/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/mips/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -174,7 +174,7 @@ no_context:
  */
 out_of_memory:
  up_read(&mm->mmap_sem);
```

```
- if (is_init(tsk)) {
+ if (is_global_init(tsk)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/powerpc/kernel/traps.c
===================================================================
--- lx26-21-mm2.orig/arch/powerpc/kernel/traps.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/powerpc/kernel/traps.c 2007-05-22 16:59:46.000000000 -0700
@@ -190,7 +190,7 @@ void _exception(int signr, struct pt_reg
   * generate the same exception over and over again and we get
   * nowhere.  Better to kill it and let the kernel panic.
   */
- if (is_init(current)) {
+ if (is_global_init(current)) {
   __sighandler_t handler;

   spin_lock_irq(&current->sighand->siglock);
Index: lx26-21-mm2/arch/powerpc/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/powerpc/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/powerpc/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -374,7 +374,7 @@ bad_area_nosemaphore:
  */
 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/powerpc/platforms/pseries/ras.c
===================================================================
--- lx26-21-mm2.orig/arch/powerpc/platforms/pseries/ras.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/powerpc/platforms/pseries/ras.c 2007-05-22 16:59:46.000000000 -0700
@@ -332,7 +332,7 @@ static int recover_mce(struct pt_regs *r
      err->disposition == RTAS_DISP_NOT_RECOVERED &&
      err->target == RTAS_TARGET_MEMORY &&
      err->type == RTAS_TYPE_ECC_UNCORR &&
-     !(current->pid == 0 || is_init(current))) {
+     !(current->pid == 0 || is_global_init(current))) {
   /* Kill off a user process with an ECC error */
   printk(KERN_ERR "MCE: uncorrectable ecc error for pid %d\n",
          current->pid);
Index: lx26-21-mm2/arch/ppc/kernel/traps.c
===================================================================
--- lx26-21-mm2.orig/arch/ppc/kernel/traps.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/ppc/kernel/traps.c 2007-05-22 16:59:46.000000000 -0700
```

```
@@ -120,7 +120,7 @@ void _exception(int signr, struct pt_reg
   * generate the same exception over and over again and we get
   * nowhere.  Better to kill it and let the kernel panic.
   */
- if (is_init(current)) {
+ if (is_global_init(current)) {
   __sighandler_t handler;

   spin_lock_irq(&current->sighand->siglock);
Index: lx26-21-mm2/arch/ppc/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/ppc/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/ppc/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -291,7 +291,7 @@ bad_area:
  */
 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/s390/lib/uaccess_pt.c
===================================================================
--- lx26-21-mm2.orig/arch/s390/lib/uaccess_pt.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/s390/lib/uaccess_pt.c 2007-05-22 16:59:46.000000000 -0700
@@ -65,7 +65,7 @@ out:

 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/s390/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/s390/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/s390/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -211,7 +211,7 @@ static int do_out_of_memory(struct pt_re
  struct mm_struct *mm = tsk->mm;

  up_read(&mm->mmap_sem);
- if (is_init(tsk)) {
+ if (is_global_init(tsk)) {
   yield();
   down_read(&mm->mmap_sem);
   return 1;
```

```
Index: lx26-21-mm2/arch/sh/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/sh/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/sh/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -233,7 +233,7 @@ no_context:
  */
 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/sh64/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/sh64/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/sh64/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -276,7 +276,7 @@ bad_area:
    show_regs(regs);
 #endif
   }
-  if (is_init(tsk)) {
+  if (is_global_init(tsk)) {
    panic("INIT had user mode bad_area\n");
   }
   tsk->thread.address = address;
@@ -318,14 +318,14 @@ no_context:
  * us unable to handle the page fault gracefully.
  */
 out_of_memory:
- if (is_init(current)) {
+ if (is_global_init(current)) {
   panic("INIT out of memory\n");
   yield();
   goto survive;
  }
  printk("fault:Out of memory\n");
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/arch/um/kernel/trap.c
===================================================================
--- lx26-21-mm2.orig/arch/um/kernel/trap.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/um/kernel/trap.c 2007-05-22 16:59:46.000000000 -0700
@@ -120,7 +120,7 @@ out_nosemaphore:
```

```
  * us unable to handle the page fault gracefully.
  */
 out_of_memory:
- if (is_init(current)) {
+ if (is_global_init(current)) {
   up_read(&mm->mmap_sem);
   yield();
   down_read(&mm->mmap_sem);
Index: lx26-21-mm2/arch/x86_64/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/x86_64/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/x86_64/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -223,7 +223,7 @@ static int is_errata93(struct pt_regs *r

 int unhandled_signal(struct task_struct *tsk, int sig)
 {
- if (is_init(tsk))
+ if (is_global_init(tsk))
   return 1;
  if (tsk->ptrace & PT_PTRACED)
   return 0;
@@ -557,7 +557,7 @@ no_context:
  */
 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   goto again;
  }
Index: lx26-21-mm2/arch/xtensa/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/xtensa/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/xtensa/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
@@ -144,7 +144,7 @@ bad_area:
   */
 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
Index: lx26-21-mm2/drivers/char/sysrq.c
===================================================================
--- lx26-21-mm2.orig/drivers/char/sysrq.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/drivers/char/sysrq.c 2007-05-22 16:59:46.000000000 -0700
@@ -250,7 +250,7 @@ static void send_sig_all(int sig)
```

```
  struct task_struct *p;

  for_each_process(p) {
-  if (p->mm && !is_init(p))
+  if (p->mm && !is_global_init(p))
     /* Not swapper, init nor kernel thread */
     force_sig(sig, p);
  }
Index: lx26-21-mm2/include/linux/sched.h
===================================================================
--- lx26-21-mm2.orig/include/linux/sched.h 2007-05-22 16:59:44.000000000 -0700
+++ lx26-21-mm2/include/linux/sched.h 2007-05-22 16:59:46.000000000 -0700
@@ -1171,16 +1171,9 @@ static inline int pid_alive(struct task_
  return p->pids[PIDTYPE_PID].pid != NULL;
 }

-/**
- * is_init - check if a task structure is init
- * @tsk: Task structure to be checked.
- *
- * Check if a task structure is the first user space task the kernel created.
- */
-static inline int is_init(struct task_struct *tsk)
-{
- return tsk->pid == 1;
-}
+struct pid_namespace;
+extern int is_global_init(struct task_struct *tsk);
+extern int is_container_init(struct task_struct *tsk);

 extern struct pid *cad_pid;

Index: lx26-21-mm2/kernel/capability.c
===================================================================
--- lx26-21-mm2.orig/kernel/capability.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/kernel/capability.c 2007-05-22 16:59:46.000000000 -0700
@@ -12,6 +12,7 @@
 #include <linux/module.h>
 #include <linux/security.h>
 #include <linux/syscalls.h>
+#include <linux/pid_namespace.h>
 #include <asm/uaccess.h>

 unsigned securebits = SECUREBITS_DEFAULT; /* systemwide security settings */
@@ -135,7 +136,7 @@ static inline int cap_set_all(kernel_cap
      int found = 0;

      do_each_thread(g, target) {
```

```
-             if (target == current || is_init(target))
+              if (target == current || is_container_init(target))
                      continue;
              found = 1;
       if (security_capset_check(target, effective, inheritable,
Index: lx26-21-mm2/kernel/exit.c
===================================================================
--- lx26-21-mm2.orig/kernel/exit.c 2007-05-22 16:59:42.000000000 -0700
+++ lx26-21-mm2/kernel/exit.c 2007-05-22 16:59:46.000000000 -0700
@@ -230,7 +230,7 @@ static int will_become_orphaned_pgrp(str
  do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
   if (p == ignored_task
     || p->exit_state
-    || is_init(p->real_parent))
+    || is_global_init(p->real_parent))
    continue;
   if (task_pgrp(p->real_parent) != pgrp &&
       task_session(p->real_parent) == task_session(p)) {
Index: lx26-21-mm2/kernel/kexec.c
===================================================================
--- lx26-21-mm2.orig/kernel/kexec.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/kernel/kexec.c 2007-05-22 16:59:46.000000000 -0700
@@ -42,7 +42,7 @@ struct resource crashk_res = {

 int kexec_should_crash(struct task_struct *p)
 {
- if (in_interrupt() || !p->pid || is_init(p) || panic_on_oops)
+ if (in_interrupt() || !p->pid || is_global_init(p) || panic_on_oops)
   return 1;
  return 0;
 }
Index: lx26-21-mm2/kernel/sysctl.c
===================================================================
--- lx26-21-mm2.orig/kernel/sysctl.c 2007-05-22 16:59:41.000000000 -0700
+++ lx26-21-mm2/kernel/sysctl.c 2007-05-22 16:59:46.000000000 -0700
@@ -1730,7 +1730,7 @@ int proc_dointvec_bset(ctl_table *table,
   return -EPERM;
  }

- op = is_init(current) ? OP_SET : OP_AND;
+ op = is_global_init(current) ? OP_SET : OP_AND;
  return do_proc_dointvec(table,write,filp,buffer,lenp,ppos,
     do_proc_dointvec_bset_conv,&op);
 }
Index: lx26-21-mm2/mm/oom_kill.c
===================================================================
--- lx26-21-mm2.orig/mm/oom_kill.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/mm/oom_kill.c 2007-05-22 16:59:46.000000000 -0700
```

```
@@ -222,7 +222,7 @@ static struct task_struct *select_bad_pr
   if (!p->mm)
    continue;
   /* skip the init task */
-  if (is_init(p))
+  if (is_global_init(p))
    continue;

   /*
@@ -275,7 +275,7 @@ static struct task_struct *select_bad_pr
  */
 static void __oom_kill_task(struct task_struct *p, int verbose)
 {
- if (is_init(p)) {
+ if (is_global_init(p)) {
   WARN_ON(1);
   printk(KERN_WARNING "tried to kill init!\n");
   return;
Index: lx26-21-mm2/security/commoncap.c
===================================================================
--- lx26-21-mm2.orig/security/commoncap.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/security/commoncap.c 2007-05-22 16:59:46.000000000 -0700
@@ -306,7 +306,7 @@ void cap_bprm_apply_creds (struct linux_
  /* For init, we want to retain the capabilities set
   * in the init_task struct. Thus we skip the usual
   * capability rules */
- if (!is_init(current)) {
+ if (!is_global_init(current)) {
   current->cap_permitted = new_permitted;
   current->cap_effective =
       cap_intersect (new_permitted, bprm->cap_effective);
Index: lx26-21-mm2/arch/avr32/kernel/traps.c
===================================================================
--- lx26-21-mm2.orig/arch/avr32/kernel/traps.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/avr32/kernel/traps.c 2007-05-22 16:59:46.000000000 -0700
@@ -88,7 +88,7 @@ void _exception(long signr, struct pt_re
   * generate the same exception over and over again and we get
   * nowhere.  Better to kill it and let the kernel panic.
   */
- if (is_init(current)) {
+ if (is_global_init(current)) {
   __sighandler_t handler;

   spin_lock_irq(&current->sighand->siglock);
Index: lx26-21-mm2/arch/avr32/mm/fault.c
===================================================================
--- lx26-21-mm2.orig/arch/avr32/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
+++ lx26-21-mm2/arch/avr32/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
```

```
@@ -173,7 +173,7 @@ bad_area:
   if (exception_trace)
    printk("%s%s[%d]: segfault at %08lx pc %08lx "
           "sp %08lx ecr %lu\n",
-          is_init(tsk) ? KERN_EMERG : KERN_INFO,
+          is_global_init(tsk) ? KERN_EMERG : KERN_INFO,
           tsk->comm, tsk->pid, address, regs->pc,
           regs->sp, ecr);
   _exception(SIGSEGV, regs, code, address);
@@ -222,7 +222,7 @@ no_context:
   */
 out_of_memory:
  up_read(&mm->mmap_sem);
- if (is_init(current)) {
+ if (is_global_init(current)) {
   yield();
   down_read(&mm->mmap_sem);
   goto survive;
@@ -244,7 +244,7 @@ do_sigbus:
  if (exception_trace)
   printk("%s%s[%d]: bus error at %08lx pc %08lx "
          "sp %08lx ecr %lu\n",
-         is_init(tsk) ? KERN_EMERG : KERN_INFO,
+         is_global_init(tsk) ? KERN_EMERG : KERN_INFO,
          tsk->comm, tsk->pid, address, regs->pc,
          regs->sp, ecr);

Index: lx26-21-mm2/kernel/pid.c
===================================================================
--- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:34.000000000 -0700
+++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
@@ -71,6 +71,27 @@ struct pid_namespace init_pid_ns = {
  .child_reaper = &init_task
 };

+
+/**
+ * is_global_init - check if a task structure is init
+ * @tsk: Task structure to be checked.
+ *
+ * Check if a task structure is the first user space task the kernel created.
+ */
+int is_global_init(struct task_struct *tsk)
+{
+ return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);
+}
+
+/*
```

```
+ * is_container_init:
+ * check whether in the task is init in it's own pid namespace.
+ */
+int is_container_init(struct task_struct *tsk)
+{
+ return tsk->pid == 1;
+}
+
 /*
  * Note: disable interrupts while the pidmap_lock is held as an
  * interrupt might come in and do read_lock(&tasklist_lock).
```

_____

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

---

Subject: [RFC][PATCH 07/16] Move alloc_pid call to copy_process
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:11:24 GMT
View Forum Message <> Reply to Message

Subject: Move alloc_pid call to copy_process

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

Move alloc_pid() into copy_process(). This will keep all pid and pid namespace code together and simplify error handling when we support multiple pid namespaces.

Changelog:
 - [Eric Biederman] Move the check of copy_process_type to alloc_pid()/ free_pid() and to avoid clutter in copy_process().

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 include/linux/pid.h | 7 ++++++-
 kernel/fork.c | 21 +++++++++++++++++++--------
 kernel/pid.c | 10 +++++++++-
 3 files changed, 28 insertions(+), 10 deletions(-)

Index: lx26-21-mm2/include/linux/pid.h
==========
--- lx26-21-mm2.orig/include/linux/pid.h 2007-05-22 16:59:40.000000000 -0700
+++ lx26-21-mm2/include/linux/pid.h 2007-05-22 17:06:48.000000000 -0700
@@ -3,6 +3,11 @@

 #include <linux/rcupdate.h>

```
+enum copy_process_type {
+ COPY_NON_IDLE_PROCESS,
+ COPY_IDLE_PROCESS,
+};
+
 enum pid_type
 {
  PIDTYPE_PID,
@@ -95,7 +100,7 @@ extern struct pid *FASTCALL(find_pid(int
 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(enum copy_process_type);
 extern void FASTCALL(free_pid(struct pid *pid));

 static inline pid_t pid_to_nr(struct pid *pid)
Index: lx26-21-mm2/kernel/fork.c
===================================================================
--- lx26-21-mm2.orig/kernel/fork.c 2007-05-22 16:59:41.000000000 -0700
+++ lx26-21-mm2/kernel/fork.c 2007-05-22 17:06:48.000000000 -0700
@@ -961,10 +961,11 @@ static struct task_struct *copy_process(
      unsigned long stack_size,
      int __user *parent_tidptr,
      int __user *child_tidptr,
-     struct pid *pid)
+     enum copy_process_type copy_src)
 {
  int retval;
  struct task_struct *p = NULL;
+ struct pid *pid;

  if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
   return ERR_PTR(-EINVAL);
@@ -1025,6 +1026,10 @@ static struct task_struct *copy_process(
  if (p->binfmt && !try_module_get(p->binfmt->module))
   goto bad_fork_cleanup_put_domain;

+ pid = alloc_pid(copy_src);
+ 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);
@@ -1305,6 +1310,8 @@ bad_fork_cleanup_cpuset:
 #endif
  cpuset_exit(p);
```

```
  delayacct_tsk_free(p);
+ free_pid(pid);
+bad_fork_put_binfmt_module:
  if (p->binfmt)
   module_put(p->binfmt->module);
 bad_fork_cleanup_put_domain:
@@ -1331,7 +1338,7 @@ struct task_struct * __cpuinit fork_idle
  struct pt_regs regs;

  task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL, NULL,
-    &init_struct_pid);
+    COPY_IDLE_PROCESS);
  if (!IS_ERR(task))
   init_idle(task, cpu);

@@ -1369,19 +1376,16 @@ long do_fork(unsigned long clone_flags,
 {
  struct task_struct *p;
  int trace = 0;
- struct pid *pid = alloc_pid();
  long nr;

- if (!pid)
-  return -EAGAIN;
- nr = pid->nr;
  if (unlikely(current->ptrace)) {
   trace = fork_traceflag (clone_flags);
   if (trace)
    clone_flags |= CLONE_PTRACE;
  }

- p = copy_process(clone_flags, stack_start, regs, stack_size, parent_tidptr, child_tidptr, pid);
+ p = copy_process(clone_flags, stack_start, regs, stack_size,
+    parent_tidptr, child_tidptr, COPY_NON_IDLE_PROCESS);
  /*
   * Do this prior waking up the new thread - the thread pointer
   * might get invalid after that point, if the thread exits quickly.
@@ -1389,6 +1393,8 @@ long do_fork(unsigned long clone_flags,
  if (!IS_ERR(p)) {
   struct completion vfork;

+  nr = pid_to_nr(task_pid(p));
+
   if (clone_flags & CLONE_VFORK) {
    p->vfork_done = &vfork;
    init_completion(&vfork);
@@ -1422,7 +1428,6 @@ long do_fork(unsigned long clone_flags,
    }
```

```
   }
  } else {
-  free_pid(pid);
   nr = PTR_ERR(p);
  }
  return nr;
Index: lx26-21-mm2/kernel/pid.c
===================================================================
--- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
+++ lx26-21-mm2/kernel/pid.c 2007-05-22 17:06:48.000000000 -0700
@@ -216,6 +216,10 @@ fastcall void free_pid(struct pid *pid)
  /* We can be called with write_lock_irq(&tasklist_lock) held */
  unsigned long flags;

+ /* check this here to keep copy_process() cleaner */
+ if (unlikely(pid == &init_struct_pid))
+  return;
+
  spin_lock_irqsave(&pidmap_lock, flags);
  hlist_del_rcu(&pid->pid_chain);
  spin_unlock_irqrestore(&pidmap_lock, flags);
@@ -224,12 +228,16 @@ fastcall void free_pid(struct pid *pid)
  call_rcu(&pid->rcu, delayed_put_pid);
 }

-struct pid *alloc_pid(void)
+struct pid *alloc_pid(enum copy_process_type copy_src)
 {
  struct pid *pid;
  enum pid_type type;
  int nr = -1;

+ /* check this here to keep copy_process() cleaner */
+ if (unlikely(copy_src == COPY_IDLE_PROCESS))
+  return &init_struct_pid;
+
  pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
  if (!pid)
   goto out;
```

_____

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

---

Subject: [RFC][PATCH 08/16] Define/use pid->upid_list list.
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:11:43 GMT

Subject: Define/use pid->upid_list list.

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

With multiple pid namespaces, a process would be known by several pid_t values, one in each pid namespace. To represent this, we introduce a 'struct upid' which associates a single pid_t value with a single pid namespace.

We then replace the pid->nr field in 'struct pid' with a list of struct upid' entries (referred to as the pid->upid_list list). This list represents the multiple pid_t values of the process, one in each namespace.

The struct upid also replaces 'struct pid' in the pid_hash table to enable us to find processes given a pid_t from any namespace (i.e we find 'struct upid' for a given pid_t and from the 'struct upid', we find the 'struct pid' of the process)

We finally reimplement find_pid() and pid_to_nr() to use pid->upid_list and remove unused fields from 'struct pid'.

Changelog:
 2.6.21-mm2-pidns3:
 - 'struct upid' used to be called 'struct pid_nr' and a list of these
    were hanging off of 'struct pid'. So, we renamed 'struct pid_nr'
    and now hold them in a statically sized array in 'struct pid' since
    the number of 'struct upid's for a process is known at process-
    creation time.

 2.6.21-rc3-mm2:

 - [Eric Biederman] Combine all logical changes into one patch
 - [Eric Biederman] Implement __pid_nr(pid_ns, pid) for use in procfs.
   (now called pid_to_nr_in_ns()).
 - [Serge Hallyn]: Remove (!pid_nr) check in free_pid_nr()

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 fs/proc/array.c | 30 ++++--
 fs/proc/base.c | 9 +-
 include/linux/init_task.h | 13 ++
 include/linux/pid.h | 35 +++++--
 include/linux/pid_namespace.h | 12 ++
 kernel/fork.c | 2
 kernel/pid.c | 187 ++++++++++++++++++++++++++++++++++++-------

```
 7 files changed, 232 insertions(+), 56 deletions(-)

Index: lx26-21-mm2/include/linux/pid.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid.h 2007-05-22 17:06:48.000000000 -0700
+++ lx26-21-mm2/include/linux/pid.h 2007-05-22 17:06:54.000000000 -0700
@@ -16,6 +16,25 @@ enum pid_type
  PIDTYPE_MAX
 };

+struct pid_namespace;
+
+/*
+ * A struct upid holds a process identifier (or pid->nr) for a given
+ * pid namespace.
+ *
+ * A list of 'struct upid' entries is stored in the struct pid. This list
+ * is used to get the process identifier associated with the pid
+ * namespace it is being seen from.
+ */
+struct upid
+{
+ /* Try to keep pid_chain in the same cacheline as nr for find_pid */
+ struct hlist_node pid_chain;  /* link hash collisions on pid_hash */
+ int nr;     /* user space pid number */
+ struct pid_namespace *pid_ns;  /* pid namespace in which nr is valid */
+ struct pid *pid;   /* back to task's unique kernel pid */
+};
+
 /*
  * What is struct pid?
  *
@@ -48,11 +67,10 @@ enum pid_type
 struct pid
 {
  atomic_t count;
- /* Try to keep pid_chain in the same cacheline as nr for find_pid */
- int nr;
- struct hlist_node pid_chain;
  /* lists of tasks that use this pid */
  struct hlist_head tasks[PIDTYPE_MAX];
+ int num_upids;
+ struct upid *upid_list;
  struct rcu_head rcu;
 };

@@ -100,16 +118,11 @@ extern struct pid *FASTCALL(find_pid(int
 extern struct pid *find_get_pid(int nr);
```

```
 extern struct pid *find_ge_pid(int nr);

-extern struct pid *alloc_pid(enum copy_process_type);
+extern struct pid *dup_struct_pid(enum copy_process_type);
 extern void FASTCALL(free_pid(struct pid *pid));

-static inline pid_t pid_to_nr(struct pid *pid)
-{
- pid_t nr = 0;
- if (pid)
-  nr = pid->nr;
- return nr;
-}
+extern pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid);
+extern pid_t pid_to_nr(struct pid *pid);

 #define do_each_pid_task(pid, type, task)    \
  do {        \
Index: lx26-21-mm2/include/linux/init_task.h
===================================================================
--- lx26-21-mm2.orig/include/linux/init_task.h 2007-05-22 17:06:48.000000000 -0700
+++ lx26-21-mm2/include/linux/init_task.h 2007-05-22 17:06:54.000000000 -0700
@@ -89,16 +89,23 @@ extern struct nsproxy init_nsproxy;

 extern struct group_info init_groups;

-#define INIT_STRUCT_PID {      \
- .count   = ATOMIC_INIT(1),    \
- .nr  = 0,       \
+#define INIT_STRUCT_UPID {      \
+ .nr  = 0,      \
  /* Don't put this struct pid in pid_hash */   \
  .pid_chain = { .next = NULL, .pprev = NULL },  \
+ .pid_ns  = &init_pid_ns,     \
+ .pid  = &init_struct_pid,    \
+}
+
+#define INIT_STRUCT_PID {      \
+ .count   = ATOMIC_INIT(1),    \
  .tasks  = {      \
   { .first = &init_task.pids[PIDTYPE_PID].node },  \
   { .first = &init_task.pids[PIDTYPE_PGID].node }, \
   { .first = &init_task.pids[PIDTYPE_SID].node },  \
  },        \
+ .num_upids = 1,      \
+ .upid_list  = &init_struct_upid,    \
  .rcu  = RCU_HEAD_INIT,    \
 }
```

```
Index: lx26-21-mm2/kernel/pid.c
===================================================================
--- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 17:06:48.000000000 -0700
+++ lx26-21-mm2/kernel/pid.c 2007-05-22 17:06:54.000000000 -0700
@@ -33,6 +33,7 @@
 static struct hlist_head *pid_hash;
 static int pidhash_shift;
 static struct kmem_cache *pid_cachep;
+struct upid init_struct_upid = INIT_STRUCT_UPID;
 struct pid init_struct_pid = INIT_STRUCT_PID;

 int pid_max = PID_MAX_DEFAULT;
@@ -195,13 +196,104 @@ static int next_pidmap(struct pid_namesp
  return -1;
 }

+static void clear_upid(struct upid *upid)
+{
+ /* We can be called with write_lock_irq(&tasklist_lock) held */
+ unsigned long flags;
+
+ free_pidmap(upid->pid_ns, upid->nr);
+
+ spin_lock_irqsave(&pidmap_lock, flags);
+ hlist_del_rcu(&upid->pid_chain);
+ spin_unlock_irqrestore(&pidmap_lock, flags);
+
+ put_pid_ns(upid->pid_ns);
+}
+
+static int init_upid(struct upid *upid, struct pid *pid,
+  struct pid_namespace *pid_ns)
+{
+ int nr;
+
+ nr = alloc_pidmap(pid_ns);
+ if (nr < 0)
+  return nr;
+
+ upid->pid_ns = pid_ns;
+ get_pid_ns(pid_ns);
+ upid->nr = nr;
+
+ /*
+  * The struct pid and list of struct upid_list represent a process
+  * with multiple pid_t values, one in each pid namespace. The list
+  * of pid_t values of a process, represented by pid->upid_list list,
```

```
+  * never changes during the life of the process. As such, struct
+  * pid and its upid_list list maybe viewed as a single object i.e
+  * they are created/destroyed together.  So we do not need a
+  * reference to struct pid here.
+  */
+ upid->pid = pid;
+
+ INIT_HLIST_NODE(&upid->pid_chain);
+ spin_lock_irq(&pidmap_lock);
+ hlist_add_head_rcu(&upid->pid_chain, &pid_hash[pid_hashfn(nr)]);
+ spin_unlock_irq(&pidmap_lock);
+
+ return 0;
+}
+
+/*
+ * Return the pid_t by which the process @pid is known in the pid
+ * namespace @ns.
+ *
+ * Return 0 if:
+ *  - @pid is NULL (eg: procfs calls this for task_pgrp(init_task)
+ *    which is NULL).
+ *
+ *  - process does not have pid_t in the namespace @ns (eg: parent
+ *    process of a child reaper does not exist in the child namespace.
+ *    A getppid() call by the child reaper results in 0).
+ */
+pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid)
+{
+ int i;
+ struct upid *upid;
+
+ if (!pid)
+  return 0;
+
+ upid = &pid->upid_list[0];
+ for (i = 0; i < pid->num_upids; i++, upid++) {
+  if (upid->pid_ns == ns)
+   return upid->nr;
+ }
+ return 0;
+}
+EXPORT_SYMBOL_GPL(pid_to_nr_in_ns);
+
+/*
+ * Return the pid_t by which the process @pid is known in the active
+ * pid namespace of the caller.
+ *
```

```
+ * pid_to_nr() cannot be static inline if task_active_pid_ns() is
+ * inline as it would cause a circular dependency between pid.h
+ * and pid_namespace.h.
+ */
+pid_t pid_to_nr(struct pid *pid)
+{
+ return pid_to_nr_in_ns(task_active_pid_ns(current), pid);
+}
+EXPORT_SYMBOL_GPL(pid_to_nr);
+
 fastcall void put_pid(struct pid *pid)
 {
  if (!pid)
   return;
+
  if ((atomic_read(&pid->count) == 1) ||
-      atomic_dec_and_test(&pid->count))
+      atomic_dec_and_test(&pid->count)) {
   kmem_cache_free(pid_cachep, pid);
+ }
 }
 EXPORT_SYMBOL_GPL(put_pid);

@@ -213,66 +305,95 @@ 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 */
- unsigned long flags;
+ int i;
+ struct upid *upid = &pid->upid_list[0];

  /* check this here to keep copy_process() cleaner */
  if (unlikely(pid == &init_struct_pid))
   return;

- spin_lock_irqsave(&pidmap_lock, flags);
- hlist_del_rcu(&pid->pid_chain);
- spin_unlock_irqrestore(&pidmap_lock, flags);
+ /* clear any upids that we actually initialized */
+ for (i = 0; i < pid->num_upids; i++, upid++) {
+  if (upid->pid_ns)
+   clear_upid(upid);
+  else
+   break;
+ }

- free_pidmap(&init_pid_ns, pid->nr);
```

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

-struct pid *alloc_pid(enum copy_process_type copy_src)
+static struct pid *alloc_struct_pid(int num_upids)
 {
  struct pid *pid;
  enum pid_type type;
- int nr = -1;
-
- /* check this here to keep copy_process() cleaner */
- if (unlikely(copy_src == COPY_IDLE_PROCESS))
-  return &init_struct_pid;
+ struct upid *upid_list;
+ void *pid_end;

+ /* for now we only support one pid namespace */
+ BUG_ON(num_upids != 1);
  pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
  if (!pid)
-  goto out;
+  return NULL;

- nr = alloc_pidmap(task_active_pid_ns(current));
- if (nr < 0)
-  goto out_free;
+ pid_end = (void *)pid + sizeof(struct pid);
+ pid->upid_list = (struct upid *)pid_end;

  atomic_set(&pid->count, 1);
- pid->nr = nr;
+ pid->num_upids = num_upids;
+
  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)]);
- spin_unlock_irq(&pidmap_lock);
+ return pid;
+}
+
+struct pid *dup_struct_pid(enum copy_process_type copy_src)
+{
+ int rc;
+ int i;
+ int num_upids;
+ struct pid *pid;
```

```
+ struct upid *upid;
+ struct upid *parent_upid;
+ struct pid *parent_pid = task_pid(current);
+
+ /* check this here to keep copy_process() cleaner */
+ if (unlikely(copy_src == COPY_IDLE_PROCESS))
+  return &init_struct_pid;
+
+ num_upids = parent_pid->num_upids;
+
+ pid = alloc_struct_pid(num_upids);
+ if (!pid)
+  return NULL;
+
+ upid = &pid->upid_list[0];
+ parent_upid = &parent_pid->upid_list[0];
+
+ for (i = 0; i < num_upids; i++, upid++, parent_upid++) {
+  rc = init_upid(upid, pid, parent_upid->pid_ns);
+  if (rc < 0)
+   goto out_free_pid;
+ }

-out:
  return pid;

-out_free:
- kmem_cache_free(pid_cachep, pid);
- pid = NULL;
- goto out;
+out_free_pid:
+ free_pid(pid);
+ return NULL;
 }

 struct pid * fastcall find_pid(int nr)
 {
  struct hlist_node *elem;
- struct pid *pid;
+ struct upid *upid;
+ struct pid_namespace *ns = task_active_pid_ns(current);

- hlist_for_each_entry_rcu(pid, elem,
+ hlist_for_each_entry_rcu(upid, elem,
    &pid_hash[pid_hashfn(nr)], pid_chain) {
-  if (pid->nr == nr)
-   return pid;
+  if ((upid->pid_ns == ns) && (upid->nr == nr))
```

```
+   return upid->pid;
  }
  return NULL;
 }
@@ -436,10 +557,14 @@ void __init pidhash_init(void)

 void __init pidmap_init(void)
 {
+ int pid_elem_size;
+
  init_pid_ns.pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
  /* Reserve PID 0. We never call free_pidmap(0) */
  set_bit(0, init_pid_ns.pidmap[0].page);
  atomic_dec(&init_pid_ns.pidmap[0].nr_free);

- pid_cachep = KMEM_CACHE(pid, SLAB_PANIC);
+ pid_elem_size = sizeof(struct pid) + sizeof(struct upid);
+ pid_cachep = kmem_cache_create("pid+1upid", pid1_elem_size, 0,
+ SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
 }
Index: lx26-21-mm2/fs/proc/array.c
===================================================================
--- lx26-21-mm2.orig/fs/proc/array.c 2007-05-22 17:06:48.000000000 -0700
+++ lx26-21-mm2/fs/proc/array.c 2007-05-22 17:06:54.000000000 -0700
@@ -75,6 +75,7 @@
 #include <linux/cpuset.h>
 #include <linux/rcupdate.h>
 #include <linux/delayacct.h>
+#include <linux/pid_namespace.h>

 #include <asm/uaccess.h>
 #include <asm/pgtable.h>
@@ -161,8 +162,18 @@ static inline char * task_state(struct t
  struct group_info *group_info;
  int g;
  struct fdtable *fdt = NULL;
+ pid_t ppid = 0;
+ pid_t tracer_pid = 0;
+ /* TODO get pid_ns from proc mnt rather than current */
+ struct pid_namespace *ns = task_active_pid_ns(current);

  rcu_read_lock();
+
+ if (pid_alive(p)) {
+  ppid = pid_to_nr_in_ns(ns, task_parent_tgid(p));
+  tracer_pid = pid_to_nr_in_ns(ns, task_tracer_pid(p));
+ }
+
```

```
  buffer += sprintf(buffer,
   "State:\t%s\n"
   "SleepAVG:\t%lu%%\n"
@@ -174,9 +185,10 @@ static inline char * task_state(struct t
   "Gid:\t%d\t%d\t%d\t%d\n",
   get_task_state(p),
   (p->sleep_avg/1024)*100/(1020000000/1024),
-        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,
+  pid_to_nr_in_ns(ns, task_tgid(p)),
+  pid_to_nr_in_ns(ns, task_pid(p)),
+  ppid,
+  tracer_pid,
   p->uid, p->euid, p->suid, p->fsuid,
   p->gid, p->egid, p->sgid, p->fsgid);

@@ -330,6 +342,8 @@ static int do_task_stat(struct task_stru
  unsigned long rsslim = 0;
  char tcomm[sizeof(task->comm)];
  unsigned long flags;
+ /* TODO get pid_ns from proc mnt rather than current */
+ struct pid_namespace *ns = task_active_pid_ns(current);

  state = *get_task_state(task);
  vsize = eip = esp = 0;
@@ -351,7 +365,7 @@ static int do_task_stat(struct task_stru
   struct signal_struct *sig = task->signal;

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

@@ -381,9 +395,9 @@ static int do_task_stat(struct task_stru
    stime = cputime_add(stime, sig->stime);
   }

-  sid = signal_session(sig);
-  pgid = process_group(task);
-  ppid = rcu_dereference(task->real_parent)->tgid;
+  sid = pid_to_nr_in_ns(ns, task_session(task));
+  pgid = pid_to_nr_in_ns(ns, task_pgrp(task));
+  ppid = pid_to_nr_in_ns(ns, task_parent_tgid(task));

   unlock_task_sighand(task, &flags);
  }
```

```
@@ -413,7 +427,7 @@ 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,
+  pid_to_nr_in_ns(ns, task_pid(task)),
   tcomm,
   state,
   ppid,
Index: lx26-21-mm2/fs/proc/base.c
===================================================================
--- lx26-21-mm2.orig/fs/proc/base.c 2007-05-22 17:06:48.000000000 -0700
+++ lx26-21-mm2/fs/proc/base.c 2007-05-22 17:06:54.000000000 -0700
@@ -72,6 +72,7 @@
 #include <linux/audit.h>
 #include <linux/poll.h>
 #include <linux/nsproxy.h>
+#include <linux/pid_namespace.h>
 #include <linux/oom.h>
 #include "internal.h"

@@ -2136,13 +2137,15 @@ static struct task_struct *next_tgid(uns
 {
  struct task_struct *task;
  struct pid *pid;
+ /* TODO get pid_ns from proc mnt rather than current */
+ struct pid_namespace *ns = task_active_pid_ns(current);

  rcu_read_lock();
 retry:
  task = NULL;
  pid = find_ge_pid(tgid);
  if (pid) {
-  tgid = pid->nr + 1;
+  tgid = pid_to_nr_in_ns(ns, pid) + 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
@@ -2182,6 +2185,8 @@ 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;
+ /* TODO get pid_ns from proc mnt rather than current */
+ struct pid_namespace *ns = task_active_pid_ns(current);

  if (!reaper)
   goto out_no_task;
@@ -2196,7 +2201,7 @@ int proc_pid_readdir(struct file * filp,
```

```
  for (task = next_tgid(tgid);
       task;
       put_task_struct(task), task = next_tgid(tgid + 1)) {
-  tgid = task->pid;
+  tgid = pid_to_nr_in_ns(ns, task_pid(task));
   filp->f_pos = tgid + TGID_OFFSET;
   if (proc_pid_fill_cache(filp, dirent, filldir, task, tgid) < 0) {
    put_task_struct(task);
Index: lx26-21-mm2/include/linux/pid_namespace.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 17:06:48.000000000 -0700
+++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 17:06:54.000000000 -0700
@@ -15,6 +15,18 @@ struct pidmap {

 #define PIDMAP_ENTRIES         ((PID_MAX_LIMIT + 8*PAGE_SIZE - 1)/PAGE_SIZE/8)

+/*
+ * Some properties/terminology for pid namespaces:
+ *
+ * Processes currently exist only in (or belong only to) the init_pid_ns.
+ * When we introduce the ability to clone the pid namespace, a process
+ * would exist in several namespaces. Off the many namespaces that the
+ * process can exist, one namespace is special and we refer to it as the
+ * 'active pid namespace' of the process.
+ *
+ * For now, we have only one pid namespace - init_pid_ns which is the
+ * 'active pid namespace' for all processes in the system.
+ */
 struct pid_namespace {
  struct kref kref;
  struct pidmap pidmap[PIDMAP_ENTRIES];
Index: lx26-21-mm2/kernel/fork.c
===================================================================
--- lx26-21-mm2.orig/kernel/fork.c 2007-05-22 17:06:48.000000000 -0700
+++ lx26-21-mm2/kernel/fork.c 2007-05-22 17:06:54.000000000 -0700
@@ -1026,7 +1026,7 @@ static struct task_struct *copy_process(
  if (p->binfmt && !try_module_get(p->binfmt->module))
   goto bad_fork_cleanup_put_domain;

- pid = alloc_pid(copy_src);
+ pid = dup_struct_pid(copy_src);
  if (!pid)
   goto bad_fork_put_binfmt_module;
```

Subject: [RFC][PATCH 09/16] Use pid ns from pid->upid_list
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:12:36 GMT
View Forum Message <> Reply to Message

Subject: Use pid ns from pid->upid_list

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

We need to decouple pid namespace from nsproxy to allow getting and releasing
pid namespace independently from other namespaces. This is required since a
process's reference to its pid namespace must exist even after its references
to other namespaces are dropped during process exit.

With multiple pid namespaces, a process can have different pid_t values in
different pid namespaces and 'struct upid' and the pid->upid_list list
provide this association of pid_t value with pid namespace for a process.

Use pid->upid_list list to find the active pid namespace of a process and
remove nsproxy->pid_namespace.

(Review note: My description of active pid namespace is probably long-winded.
Appreciate any comments in explaining it better :-)

TODO:
- Include pid_namespace in pid_hash() so processes with same pid_t
in different namespaces are on different hash lists.

Changelog:

2.6.21-mm2-pidns3:
- 'struct upid' used to be called 'struct pid_nr' and a list of these
were hanging off of 'struct pid'. So, we renamed 'struct pid_nr'
and now hold them in a statically sized array in 'struct pid' since
the number of 'struct upid's for a process is known at process-
creation time.

[2.6.21-rc3-mm2]
- Drop support for unshare() of pid namespace which simplifies cloning
of pid namespace and reorganize several functions.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
include/linux/init_task.h | 1 -
include/linux/nsproxy.h | 2 --
include/linux/pid_namespace.h | 4 ++--
kernel/nsproxy.c | 10 ----------
kernel/pid.c | 23 ++++++++++++++++++++-------
5 files changed, 18 insertions(+), 22 deletions(-)

```
Index: lx26-21-mm2/include/linux/init_task.h
===================================================================
--- lx26-21-mm2.orig/include/linux/init_task.h 2007-05-22 16:59:49.000000000 -0700
+++ lx26-21-mm2/include/linux/init_task.h 2007-05-22 16:59:50.000000000 -0700
@@ -72,7 +72,6 @@

 extern struct nsproxy init_nsproxy;
 #define INIT_NSPROXY(nsproxy) {      \
- .pid_ns  = &init_pid_ns,     \
  .count  = ATOMIC_INIT(1),    \
  .nslock  = __SPIN_LOCK_UNLOCKED(nsproxy.nslock),  \
  .uts_ns  = &init_uts_ns,     \
Index: lx26-21-mm2/include/linux/nsproxy.h
===================================================================
--- lx26-21-mm2.orig/include/linux/nsproxy.h 2007-05-22 16:58:37.000000000 -0700
+++ lx26-21-mm2/include/linux/nsproxy.h 2007-05-22 16:59:50.000000000 -0700
@@ -7,7 +7,6 @@
 struct mnt_namespace;
 struct uts_namespace;
 struct ipc_namespace;
-struct pid_namespace;

 /*
  * A structure to contain pointers to all per-process
@@ -27,7 +26,6 @@ struct nsproxy {
  struct uts_namespace *uts_ns;
  struct ipc_namespace *ipc_ns;
  struct mnt_namespace *mnt_ns;
- struct pid_namespace *pid_ns;
 };
 extern struct nsproxy init_nsproxy;

Index: lx26-21-mm2/include/linux/pid_namespace.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:49.000000000 -0700
+++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 16:59:50.000000000 -0700
@@ -41,8 +41,8 @@ static inline void get_pid_ns(struct pid
  kref_get(&ns->kref);
 }

-extern struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *ns);
 extern void free_pid_ns(struct kref *kref);
+extern struct pid_namespace *pid_active_pid_ns(struct pid *pid);

 static inline void put_pid_ns(struct pid_namespace *ns)
 {
@@ -51,7 +51,7 @@ static inline void put_pid_ns(struct pid
```

```
 static inline struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
 {
- return tsk->nsproxy->pid_ns;
+ return pid_active_pid_ns(task_pid(tsk));
 }

 static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
Index: lx26-21-mm2/kernel/nsproxy.c
===================================================================
--- lx26-21-mm2.orig/kernel/nsproxy.c 2007-05-22 16:58:37.000000000 -0700
+++ lx26-21-mm2/kernel/nsproxy.c 2007-05-22 16:59:50.000000000 -0700
@@ -19,7 +19,6 @@
 #include <linux/init_task.h>
 #include <linux/mnt_namespace.h>
 #include <linux/utsname.h>
-#include <linux/pid_namespace.h>

 struct nsproxy init_nsproxy = INIT_NSPROXY(init_nsproxy);

@@ -75,15 +74,8 @@ static struct nsproxy *create_new_namesp
  if (IS_ERR(new_nsp->ipc_ns))
   goto out_ipc;

- new_nsp->pid_ns = copy_pid_ns(flags, tsk->nsproxy->pid_ns);
- if (IS_ERR(new_nsp->pid_ns))
-  goto out_pid;
-
  return new_nsp;

-out_pid:
- if (new_nsp->ipc_ns)
-  put_ipc_ns(new_nsp->ipc_ns);
 out_ipc:
  if (new_nsp->uts_ns)
   put_uts_ns(new_nsp->uts_ns);
@@ -138,8 +130,6 @@ void free_nsproxy(struct nsproxy *ns)
   put_uts_ns(ns->uts_ns);
  if (ns->ipc_ns)
   put_ipc_ns(ns->ipc_ns);
- if (ns->pid_ns)
-  put_pid_ns(ns->pid_ns);
  kfree(ns);
 }

Index: lx26-21-mm2/kernel/pid.c
===================================================================
--- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:49.000000000 -0700
```

```
+++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:50.000000000 -0700
@@ -242,6 +242,22 @@ static int init_upid(struct upid *upid,
  return 0;
 }

+static struct upid *pid_active_upid(struct pid *pid)
+{
+ return &pid->upid_list[0];
+}
+
+/*
+ * Return the active pid namespace of the process @pid.
+ *
+ * Note: At present, there is only one pid namespace (init_pid_ns).
+ */
+struct pid_namespace *pid_active_pid_ns(struct pid *pid)
+{
+ return pid_active_upid(pid)->pid_ns;
+}
+EXPORT_SYMBOL_GPL(pid_active_pid_ns);
+
 /*
  * Return the pid_t by which the process @pid is known in the pid
  * namespace @ns.
@@ -515,13 +531,6 @@ struct pid *find_ge_pid(int nr)
 }
 EXPORT_SYMBOL_GPL(find_get_pid);

-struct pid_namespace *copy_pid_ns(int flags, struct pid_namespace *old_ns)
-{
- BUG_ON(!old_ns);
- get_pid_ns(old_ns);
- return old_ns;
-}
-
 void free_pid_ns(struct kref *kref)
 {
  struct pid_namespace *ns;
```

Subject: [RFC][PATCH 10/16] Define CLONE_NEWPID flag
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:13:03 GMT
View Forum Message <> Reply to Message

Subject: Define CLONE_NEWPID flag

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

This was discussed on containers and we thought it would be useful
to reserve this flag.

TODO: Merge this with the next patch ?

Define CLONE_NEWPID flag that will be used to clone pid namespaces.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 include/linux/sched.h | 1 +
 1 file changed, 1 insertion(+)

Index: lx26-21-mm2/include/linux/sched.h
===
--- lx26-21-mm2.orig/include/linux/sched.h 2007-05-22 16:59:46.000000000 -0700
+++ lx26-21-mm2/include/linux/sched.h 2007-05-22 16:59:51.000000000 -0700
@@ -26,6 +26,7 @@
 #define CLONE_STOPPED 0x02000000 /* Start in stopped state */
 #define CLONE_NEWUTS 0x04000000 /* New utsname group? */
 #define CLONE_NEWIPC 0x08000000 /* New ipcs */
+#define CLONE_NEWPID 0x10000000   /* New pid namespace */

 /*
  * Scheduling policies

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

---

Subject: [RFC][PATCH 11/16] Enable cloning pid namespace
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:13:22 GMT
View Forum Message <> Reply to Message

Subject: Enable cloning pid namespace

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

When clone() is invoked with CLONE_NEWPID, create a new pid namespace
and then create a new struct pid for the new process. Allocate pid_t's
for the new process in the new pid namespace and all ancestor pid
namespaces. Make the newly cloned process the session and process group

leader.

Since the active pid namespace is special and expected to be the first
entry in pid->upid_list, preserve the order of pid namespaces when
cloning without CLONE_NEWPID.

TODO (partial list:)

 - Identify clone flags that should not be specified with CLONE_NEWPID
        and return -EINVAL from copy_process(), if they are specified. (eg:
   CLONE_THREAD|CLONE_NEWPID ?)
 - Add a privilege check for CLONE_NEWPID

Changelog:

 2.6.21-mm2-pidns3:
       - 'struct upid' used to be called 'struct pid_nr' and a list of these
         were hanging off of 'struct pid'. So, we renamed 'struct pid_nr'
         and now hold them in a statically sized array in 'struct pid' since
         the number of 'struct upid's for a process is known at process-
         creation time

 2.6.21-mm2:
 - [Serge Hallyn] Terminate other processes in pid ns when reaper is
   exiting.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 include/linux/pid.h | 3
 include/linux/pid_namespace.h | 5 -
 init/Kconfig | 9 +
 kernel/exit.c | 14 ++
 kernel/fork.c | 17 ++-
 kernel/pid.c | 209 +++++++++++++++++++++++++++++++++++++++----
 6 files changed, 227 insertions(+), 30 deletions(-)

Index: lx26-21-mm2/kernel/pid.c
===
--- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:50.000000000 -0700
+++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:52.000000000 -0700
@@ -32,6 +32,7 @@
 #define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
 static struct hlist_head *pid_hash;
 static int pidhash_shift;
+static struct kmem_cache *pid1_cachep;
 static struct kmem_cache *pid_cachep;
 struct upid init_struct_upid = INIT_STRUCT_UPID;
 struct pid init_struct_pid = INIT_STRUCT_PID;

```
@@ -250,7 +251,12 @@ static struct upid *pid_active_upid(stru
 /*
  * Return the active pid namespace of the process @pid.
  *
- * Note: At present, there is only one pid namespace (init_pid_ns).
+ * Note:
+ * To avoid having to use an extra pointer in struct pid to keep track
+ * of active pid namespace, dup_struct_pid() maintains the order of
+ * entries in 'pid->upid_list' such that the youngest (or the 'active')
+ * pid namespace is the first entry and oldest (init_pid_ns) is the last
+ * entry in the list.
  */
 struct pid_namespace *pid_active_pid_ns(struct pid *pid)
 {
@@ -259,6 +265,64 @@ struct pid_namespace *pid_active_pid_ns(
 EXPORT_SYMBOL_GPL(pid_active_pid_ns);

 /*
+ * Return the parent pid_namespace of the active pid namespace of @tsk.
+ *
+ * Note:
+ *  Refer to function header of pid_active_pid_ns() for information on
+ *  the order of entries in pid->upid_list. Based on the order, the parent
+ *  pid namespace of the active pid namespace of @tsk is just the second
+ *  entry in the process's pid->upid_list.
+ *
+ *  Parent pid namespace of init_pid_ns is init_pid_ns itself.
+ */
+static struct pid_namespace *task_active_pid_ns_parent(struct task_struct *tsk)
+{
+ int idx = 0;
+ struct pid *pid = task_pid(tsk);
+
+ if (pid->num_upids > 1)
+  idx++;
+
+ return pid->upid_list[idx].pid_ns;
+}
+
+/*
+ * Return the child reaper of @tsk.
+ *
+ * Normally the child reaper of @tsk is simply the child reaper
+ * the active pid namespace of @tsk.
+ *
+ * But if @tsk is itself child reaper of a namespace, NS1, its child
+ * reaper depends on the caller.  If someone from an ancestor namespace
+ * or, if the reaper himself is asking, return the reaper of our parent
```

```
+ * namespace.
+ *
+ * If someone from namespace NS1 (other than reaper himself) is asking,
+ * return reaper of NS1.
+ */
+struct task_struct *task_child_reaper(struct task_struct *tsk)
+{
+ struct pid_namespace *tsk_ns = task_active_pid_ns(tsk);
+ struct task_struct *tsk_reaper = tsk_ns->child_reaper;
+ struct pid_namespace *my_ns;
+
+ /*
+  * TODO: Check if we need a lock here. ns->child_reaper
+  *   can change in do_exit() when reaper is exiting.
+  */
+
+ if (tsk != tsk_reaper)
+  return tsk_reaper;
+
+ my_ns = task_active_pid_ns(current);
+ if (my_ns != tsk_ns || current == tsk)
+  return task_active_pid_ns_parent(tsk)->child_reaper;
+
+ return tsk_reaper;
+}
+EXPORT_SYMBOL(task_child_reaper);
+
+/*
  * Return the pid_t by which the process @pid is known in the pid
  * namespace @ns.
  *
@@ -301,15 +365,78 @@ pid_t pid_to_nr(struct pid *pid)
 }
 EXPORT_SYMBOL_GPL(pid_to_nr);

+#ifdef CONFIG_PID_NS
+static int init_ns_pidmap(struct pid_namespace *ns)
+{
+ int i;
+
+ atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
+
+ ns->pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
+ if (!ns->pidmap[0].page)
+  return -ENOMEM;
+
+ set_bit(0, ns->pidmap[0].page);
+
```

```
+ for (i = 1; i < PIDMAP_ENTRIES; i++) {
+  atomic_set(&ns->pidmap[i].nr_free, BITS_PER_PAGE);
+  ns->pidmap[i].page = NULL;
+ }
+ return 0;
+}
+
+static struct pid_namespace *alloc_pid_ns(void)
+{
+ struct pid_namespace *ns;
+ int rc;
+
+ ns = kzalloc(sizeof(struct pid_namespace), GFP_KERNEL);
+ if (!ns)
+  return NULL;
+
+ rc = init_ns_pidmap(ns);
+ if(rc) {
+  kfree(ns);
+  return NULL;
+ }
+
+ kref_init(&ns->kref);
+
+ return ns;
+}
+
+#else
+
+static int alloc_pid_ns()
+{
+ static int warned;
+
+ if (!warned) {
+  printk(KERN_INFO "WARNING: CLONE_NEWPID disabled\n");
+  warned = 1;
+ }
+ return 0;
+}
+#endif  /*CONFIG_PID_NS*/
+
+void toss_pid(struct pid *pid)
+{
+ if (pid->num_upids == 1)
+  kmem_cache_free(pid1_cachep, pid);
+ else {
+  kfree(pid->upid_list);
+  kmem_cache_free(pid_cachep, pid);
```

```
+ }
+}
+
 fastcall void put_pid(struct pid *pid)
 {
  if (!pid)
   return;

  if ((atomic_read(&pid->count) == 1) ||
-      atomic_dec_and_test(&pid->count)) {
-  kmem_cache_free(pid_cachep, pid);
- }
+      atomic_dec_and_test(&pid->count))
+  toss_pid(pid);
 }
 EXPORT_SYMBOL_GPL(put_pid);

@@ -345,15 +472,28 @@ static struct pid *alloc_struct_pid(int
  enum pid_type type;
  struct upid *upid_list;
  void *pid_end;
+ struct kmem_cache *cachep = pid1_cachep;

- /* for now we only support one pid namespace */
- BUG_ON(num_upids != 1);
- pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
+ if (num_upids > 1)
+  cachep = pid_cachep;
+
+ pid = kmem_cache_alloc(cachep, GFP_KERNEL);
  if (!pid)
   return NULL;

- pid_end = (void *)pid + sizeof(struct pid);
- pid->upid_list = (struct upid *)pid_end;
+ if (num_upids == 1) {
+  pid_end = (void *)pid + sizeof(struct pid);
+  pid->upid_list = (struct upid *)pid_end;
+ } else {
+  int upid_list_size = num_upids * sizeof(struct upid);
+
+  upid_list = kzalloc(upid_list_size, GFP_KERNEL);
+  if (!upid_list) {
+   kmem_cache_free(pid_cachep, pid);
+   return NULL;
+  }
+  pid->upid_list = upid_list;
+ }
```

```
  atomic_set(&pid->count, 1);
  pid->num_upids = num_upids;
@@ -364,7 +504,8 @@ static struct pid *alloc_struct_pid(int
  return pid;
 }

-struct pid *dup_struct_pid(enum copy_process_type copy_src)
+struct pid *dup_struct_pid(enum copy_process_type copy_src,
+  unsigned long clone_flags, struct task_struct *new_task)
 {
  int rc;
  int i;
@@ -379,20 +520,38 @@ struct pid *dup_struct_pid(enum copy_pro
   return &init_struct_pid;

  num_upids = parent_pid->num_upids;
+ if (clone_flags & CLONE_NEWPID)
+  num_upids++;

  pid = alloc_struct_pid(num_upids);
  if (!pid)
   return NULL;

  upid = &pid->upid_list[0];
+
+ if (clone_flags & CLONE_NEWPID) {
+  struct pid_namespace *new_pid_ns = alloc_pid_ns();
+
+  if (!new_pid_ns)
+   goto out_free_pid;
+
+  new_pid_ns->child_reaper = new_task;
+  rc = init_upid(upid, pid, new_pid_ns);
+  if (rc < 0)
+   goto out_free_pid;
+  upid++;
+ }
+
  parent_upid = &parent_pid->upid_list[0];

- for (i = 0; i < num_upids; i++, upid++, parent_upid++) {
+ for (i = 0; i < parent_pid->num_upids; i++, upid++, parent_upid++) {
   rc = init_upid(upid, pid, parent_upid->pid_ns);
   if (rc < 0)
    goto out_free_pid;
  }
```

```
+ new_task->pid = pid_active_upid(pid)->nr;
+
  return pid;

 out_free_pid:
@@ -533,9 +692,21 @@ EXPORT_SYMBOL_GPL(find_get_pid);

 void free_pid_ns(struct kref *kref)
 {
+ int i;
+ int nr_free;
  struct pid_namespace *ns;

  ns = container_of(kref, struct pid_namespace, kref);
+
+ BUG_ON(ns == &init_pid_ns);
+
+ for (i = 0; i < PIDMAP_ENTRIES; i++) {
+  nr_free = atomic_read(&ns->pidmap[i].nr_free);
+  BUG_ON(nr_free != BITS_PER_PAGE);
+
+  if (ns->pidmap[i].page)
+   kfree(ns->pidmap[i].page);
+ }
  kfree(ns);
 }

@@ -566,14 +737,22 @@ void __init pidhash_init(void)

 void __init pidmap_init(void)
 {
- int pid_elem_size;
+ int pid1_elem_size;

  init_pid_ns.pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
  /* Reserve PID 0. We never call free_pidmap(0) */
  set_bit(0, init_pid_ns.pidmap[0].page);
  atomic_dec(&init_pid_ns.pidmap[0].nr_free);

- pid_elem_size = sizeof(struct pid) + sizeof(struct upid);
- pid_cachep = kmem_cache_create("pid+1upid", pid1_elem_size, 0,
- SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
+ /*
+  * Cache for struct pids with more than one pid namespace
+  */
+ pid_cachep = KMEM_CACHE(pid, SLAB_PANIC);
+
+ /*
```

```
+  * Cache for struct pids with exactly one pid namespace
+  */
+ pid1_elem_size = sizeof(struct pid) + sizeof(struct upid);
+ pid1_cachep = kmem_cache_create("pid+1upid", pid1_elem_size, 0,
+    SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
 }
Index: lx26-21-mm2/include/linux/pid.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid.h 2007-05-22 16:59:49.000000000 -0700
+++ lx26-21-mm2/include/linux/pid.h 2007-05-22 16:59:52.000000000 -0700
@@ -118,7 +118,8 @@ extern struct pid *FASTCALL(find_pid(int
 extern struct pid *find_get_pid(int nr);
 extern struct pid *find_ge_pid(int nr);

-extern struct pid *dup_struct_pid(enum copy_process_type);
+extern struct pid *dup_struct_pid(enum copy_process_type,
+  unsigned long clone_flags, struct task_struct *new_task);
 extern void FASTCALL(free_pid(struct pid *pid));

 extern pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid);
Index: lx26-21-mm2/include/linux/pid_namespace.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:50.000000000 -0700
+++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 16:59:52.000000000 -0700
@@ -54,9 +54,6 @@ static inline struct pid_namespace *task
  return pid_active_pid_ns(task_pid(tsk));
 }

-static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
-{
- return task_active_pid_ns(tsk)->child_reaper;
-}
+extern struct task_struct *task_child_reaper(struct task_struct *tsk);

 #endif /* _LINUX_PID_NS_H */
Index: lx26-21-mm2/init/Kconfig
===================================================================
--- lx26-21-mm2.orig/init/Kconfig 2007-05-22 16:58:36.000000000 -0700
+++ lx26-21-mm2/init/Kconfig 2007-05-22 16:59:52.000000000 -0700
@@ -250,6 +250,15 @@ config UTS_NS
    vservers, to use uts namespaces to provide different
    uts info for different servers.  If unsure, say N.

+config PID_NS
+ depends on EXPERIMENTAL
+ bool "PID Namespaces"
+ default n
+ help
```

```
+   Support multiple PID namespaces. This allows containers, i.e.
+   vservers to use separate different PID namespaces to different
+   servers. If unsuare, say N.
+
 config AUDIT
  bool "Auditing support"
  depends on NET
Index: lx26-21-mm2/kernel/exit.c
===================================================================
--- lx26-21-mm2.orig/kernel/exit.c 2007-05-22 16:59:46.000000000 -0700
+++ lx26-21-mm2/kernel/exit.c 2007-05-22 16:59:52.000000000 -0700
@@ -866,6 +866,7 @@ fastcall NORET_TYPE void do_exit(long co
 {
  struct task_struct *tsk = current;
  int group_dead;
+ struct pid_namespace *pid_ns = task_active_pid_ns(tsk);

  profile_task_exit(tsk);

@@ -875,10 +876,15 @@ 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 == 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;
+
+ /*
+  * Note that we cannot use task_child_reaper() here because
+  * it returns reaper for parent pid namespace if tsk is itself
+  * the reaper of the active pid namespace.
+  */
+ if (unlikely(tsk == pid_ns->child_reaper)) {
+  if (pid_ns != &init_pid_ns)
+   pid_ns->child_reaper = init_pid_ns.child_reaper;
   else
    panic("Attempted to kill init!");
  }
Index: lx26-21-mm2/kernel/fork.c
===================================================================
--- lx26-21-mm2.orig/kernel/fork.c 2007-05-22 16:59:49.000000000 -0700
+++ lx26-21-mm2/kernel/fork.c 2007-05-22 16:59:52.000000000 -0700
@@ -1026,14 +1026,13 @@ static struct task_struct *copy_process(
  if (p->binfmt && !try_module_get(p->binfmt->module))
   goto bad_fork_cleanup_put_domain;

- pid = dup_struct_pid(copy_src);
```

```
+ pid = dup_struct_pid(copy_src, clone_flags, p);
  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_to_nr(pid);

  INIT_LIST_HEAD(&p->children);
  INIT_LIST_HEAD(&p->sibling);
@@ -1255,11 +1254,17 @@ static struct task_struct *copy_process(
    __ptrace_link(p, current->parent);

   if (thread_group_leader(p)) {
+   struct pid *pgrp = task_pgrp(current);
+   struct pid *session = task_session(current);
+
+   if (clone_flags & CLONE_NEWPID)
+    pgrp = session = pid;
+
    p->signal->tty = current->signal->tty;
-   p->signal->pgrp = process_group(current);
-   set_signal_session(p->signal, process_session(current));
-   attach_pid(p, PIDTYPE_PGID, task_pgrp(current));
-   attach_pid(p, PIDTYPE_SID, task_session(current));
+   p->signal->pgrp = pid_to_nr(pgrp);
+   set_signal_session(p->signal, pid_to_nr(session));
+   attach_pid(p, PIDTYPE_PGID, pgrp);
+   attach_pid(p, PIDTYPE_SID, session);

    list_add_tail_rcu(&p->tasks, &init_task.tasks);
    __get_cpu_var(process_counts)++;
```

_____

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

---

Subject: [RFC][PATCH 12/16] Terminate processes in a ns when reaper is exiting.
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:13:54 GMT
View Forum Message <> Reply to Message

Subject: Terminate processes in a ns when reaper is exiting.

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

This should actually be considered a part of the previous patch which

enables cloning of pid namespace. Its been separated out for easier
review.

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.

TODO:
 - Consider maintaining a per-pid namespace tasklist. Use that list
   to terminate processes in the namespace more efficiently. Such a
   tasklist may also be useful to freeze or checkpoint an application.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 include/linux/pid.h | 1 +
 include/linux/pid_namespace.h | 1 +
 kernel/exit.c | 5 +++--
 kernel/fork.c | 19 ++++++++++++++++++
 kernel/pid.c | 42 ++
 5 files changed, 66 insertions(+), 2 deletions(-)

Index: lx26-21-mm2/include/linux/pid.h
===
--- lx26-21-mm2.orig/include/linux/pid.h 2007-05-22 16:59:52.000000000 -0700
+++ lx26-21-mm2/include/linux/pid.h 2007-05-22 16:59:53.000000000 -0700
@@ -124,6 +124,7 @@ extern void FASTCALL(free_pid(struct pid

 extern pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid);
 extern pid_t pid_to_nr(struct pid *pid);
+extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

 #define do_each_pid_task(pid, type, task)   \
  do {   \
Index: lx26-21-mm2/include/linux/pid_namespace.h
===
--- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:52.000000000 -0700
+++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 16:59:53.000000000 -0700
@@ -32,6 +32,7 @@ struct pid_namespace {
  struct pidmap pidmap[PIDMAP_ENTRIES];
  int last_pid;
  struct task_struct *child_reaper;
+ atomic_t terminating;
 };

 extern struct pid_namespace init_pid_ns;
Index: lx26-21-mm2/kernel/exit.c
===
--- lx26-21-mm2.orig/kernel/exit.c 2007-05-22 16:59:52.000000000 -0700

```
+++ lx26-21-mm2/kernel/exit.c 2007-05-22 16:59:53.000000000 -0700
@@ -883,9 +883,10 @@ fastcall NORET_TYPE void do_exit(long co
   * the reaper of the active pid namespace.
   */
  if (unlikely(tsk == pid_ns->child_reaper)) {
-  if (pid_ns != &init_pid_ns)
+  if (pid_ns != &init_pid_ns) {
+   zap_pid_ns_processes(pid_ns);
    pid_ns->child_reaper = init_pid_ns.child_reaper;
-  else
+  } else
    panic("Attempted to kill init!");
  }

Index: lx26-21-mm2/kernel/fork.c
===================================================================
--- lx26-21-mm2.orig/kernel/fork.c 2007-05-22 16:59:52.000000000 -0700
+++ lx26-21-mm2/kernel/fork.c 2007-05-22 16:59:53.000000000 -0700
@@ -1397,6 +1397,7 @@ long do_fork(unsigned long clone_flags,
   */
  if (!IS_ERR(p)) {
   struct completion vfork;
+  struct pid_namespace *pid_ns;

   nr = pid_to_nr(task_pid(p));

@@ -1405,6 +1406,24 @@ long do_fork(unsigned long clone_flags,
    init_completion(&vfork);
   }

+  /*
+   * If our pid namespace was asked to terminate while we were
+   * being created, exit immediately.
+   *
+   * We need the check here for the case where the termination
+   * request was received after a pid_t was allocated in
+   * alloc_pid_nr() and before the process was fully created
+   * ("old enough") to receive the signal sent from do_exit().
+   *
+   * TODO: Implement a task list per pid namespace to simplify
+   * terminating the pid namespace and remove this check.
+   */
+  pid_ns = task_active_pid_ns(p);
+  if (atomic_read(&pid_ns->terminating)) {
+   sigaddset(&p->pending.signal, SIGKILL);
+   set_tsk_thread_flag(p, TIF_SIGPENDING);
+  }
+
```

```
   if ((p->ptrace & PT_PTRACED) || (clone_flags & CLONE_STOPPED)) {
    /*
     * We'll start up with an immediate SIGSTOP.
Index: lx26-21-mm2/kernel/pid.c
===================================================================
--- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:52.000000000 -0700
+++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:53.000000000 -0700
@@ -124,6 +124,9 @@ static int alloc_pidmap(struct pid_names
  int i, offset, max_scan, pid, last = pid_ns->last_pid;
  struct pidmap *map;

+ if (atomic_read(&pid_ns->terminating))
+  return -1;
+
  pid = last + 1;
  if (pid >= pid_max)
   pid = RESERVED_PIDS;
@@ -405,6 +408,39 @@ static struct pid_namespace *alloc_pid_n
  return ns;
 }

+/*
+ * When child reaper of the pid namespace @pid_ns is itself terminating,
+ * we need to terminate all processes in the pid namespace since /proc
+ * has a reference to the child reaper of the pid namespace.
+ *
+ * Send SIGKILL to all processes in the pid namespace. Set the 'terminating'
+ * flag in pid_ns to prevent any new processes from getting created in the
+ * pid namespace.
+ *
+ * Note that we will also be terminating all our child pid namespaces
+ * (if any) since we send SIGKILL their reapers as well.
+ *
+ * TODO: It maybe more efficient to maintain a list of tasks in the
+ *       pid namespace and walk that list.
+ */
+void zap_pid_ns_processes(struct pid_namespace *pid_ns)
+{
+ int nr;
+
+ atomic_set(&pid_ns->terminating, 1);
+
+ /*
+  * We know pid == 1 is terminating. Find remaining pid_ts
+  * in the namespace and terminate them.
+  */
+ nr = next_pidmap(pid_ns, 1);
+ while (nr > 0) {
```

```
+  kill_proc(nr, SIGKILL, 1);
+  nr = next_pidmap(pid_ns, nr);
+ }
+ return;
+}
+
 #else

 static int alloc_pid_ns()
@@ -417,6 +453,12 @@ static int alloc_pid_ns()
  }
  return 0;
 }
+
+void zap_pid_ns_processes(struct pid_namespace *pid_ns)
+{
+ /* Nothing to do when we don't have multiple pid namespaces */
+ return;
+}
 #endif  /*CONFIG_PID_NS*/

 void toss_pid(struct pid *pid)
```

Subject: [RFC][PATCH 13/16] Remove proc_mnt's use for killing inodes
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:14:17 GMT
View Forum Message <> Reply to Message

Subject: Remove proc_mnt's use for killing inodes

From: Dave Hansen <hansendc@us.ibm.com>

We use proc_mnt as a shortcut to find a superblock on which
to go killing /proc inodes. This will break if we ever have
more than one /proc superblock. So, use the superblock list
to go find each of the /proc sb's and kill inodes on each
superblock.

This does introduce an extra lock grab from what was there
before, but the list should be only 1 long 99% of the time,
and we don't exactly remove proc entries in hot paths. Note
that this *isn't* the path that we use to get rid of the
actual /proc pid entries. Those are a different beast.

```
Signed-off-by: Dave Hansen <haveblue@us.ibm.com>
---
 fs/proc/generic.c |   28 ++++++++++++++++++++++-----
 1 file changed, 23 insertions(+), 5 deletions(-)

Index: lx26-21-mm2/fs/proc/generic.c
===================================================================
--- lx26-21-mm2.orig/fs/proc/generic.c 2007-05-22 16:51:19.000000000 -0700
+++ lx26-21-mm2/fs/proc/generic.c 2007-05-22 16:59:55.000000000 -0700
@@ -554,13 +554,10 @@ static int proc_register(struct proc_dir
  return 0;
 }

-/*
- * Kill an inode that got unregistered..
- */
-static void proc_kill_inodes(struct proc_dir_entry *de)
+static void proc_kill_inodes_sb(struct proc_dir_entry *de,
+    struct super_block *sb)
 {
  struct list_head *p;
- struct super_block *sb = proc_mnt->mnt_sb;

  /*
   * Actually it's a partial revoke().
@@ -584,6 +581,27 @@ static void proc_kill_inodes(struct proc
  file_list_unlock();
 }

+/*
+ * Kill an inode that got unregistered..
+ */
+static void proc_kill_inodes(struct proc_dir_entry *de)
+{
+ struct list_head *l;
+ struct file_system_type *procfs;
+
+ procfs = get_fs_type("proc");
+ if (!procfs)
+  return;
+
+ spin_lock(&sb_lock);
+ list_for_each(l, &procfs->fs_supers) {
+  struct super_block *sb;
+  sb = list_entry(l, struct super_block, s_instances);
+  proc_kill_inodes_sb(de, sb);
+ }
```

```
+ spin_unlock(&sb_lock);
+}
+
 static struct proc_dir_entry *proc_create(struct proc_dir_entry **parent,
       const char *name,
       mode_t mode,
```


Subject: [RFC][PATCH 14/16] Introduce proc_mnt for pid_ns
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:14:42 GMT

View Forum Message <> Reply to Message

Subject: Introduce proc_mnt for pid_ns

From: Dave Hansen <hansendc@us.ibm.com>

The following patch completes the removal of the global proc_mnt.
It fetches the mnt on which to do dentry invalidations from the
pid_namespace in which the task appears.

For now, there is only one pid namespace in mainline so this is
straightforward. In the -lxc tree we'll have to do something
more complex. The proc_flush_task() code takes a task, and
needs to be able to find the corresponding proc superblocks on
which that tasks's /proc/<pid> directories could appear. We
can tell in which pid namespaces a task appears, so I put a
pointer from the pid namespace to the corresponding proc_mnt.

/proc currently has some special code to make sure that the root
directory gets set up correctly. It proc_mnt variable in order
to find its way to the root inode.

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

```
 fs/proc/base.c               |  32 +++++++++++++++++++++++++-
 fs/proc/inode.c              |  11 ++++++++-
 fs/proc/root.c               |  52 ++++++++++++++++++++++++++++++++--------------
 include/linux/pid_namespace.h |   1
 include/linux/proc_fs.h      |   1
 5 files changed, 75 insertions(+), 22 deletions(-)
```

Index: lx26-21-mm2/fs/proc/base.c

```
===================================================================
--- lx26-21-mm2.orig/fs/proc/base.c 2007-05-22 16:59:49.000000000 -0700
+++ lx26-21-mm2/fs/proc/base.c 2007-05-22 16:59:56.000000000 -0700
@@ -2005,9 +2005,11 @@ static const struct inode_operations pro
 };

 /**
- * proc_flush_task -  Remove dcache entries for @task from the /proc dcache.
+ * proc_flush_task_from_pid_ns -  Remove dcache entries for @task
+ *       from the /proc dcache.
  *
  * @task: task that should be flushed.
+ * @pid_ns: pid_namespace in which that task appears
  *
  * Looks in the dcache for
  * /proc/@pid
@@ -2025,11 +2027,22 @@ 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_from_pid_ns(struct task_struct *task,
+    struct pid_namespace* pid_ns)
 {
  struct dentry *dentry, *leader, *dir;
  char buf[PROC_NUMBUF];
  struct qstr name;
+ struct vfsmount *proc_mnt;
+
+ WARN_ON(!pid_ns);
+ /*
+  * It is possible that no /procs have been instantiated
+  * for this particular pid namespace.
+  */
+ if (!pid_ns->proc_mnt)
+  return;
+ proc_mnt = pid_ns->proc_mnt;

  name.name = buf;
  name.len = snprintf(buf, sizeof(buf), "%d", task->pid);
@@ -2071,6 +2084,21 @@ out:
  return;
 }

+void proc_flush_task(struct task_struct *task)
+{
+ int i;
+ struct pid *pid;
```

```
+ struct upid* upid;
+
+ pid = task_pid(task);
+ if (!pid)
+  return;
+
+ upid = &pid->upid_list[0];
+ for (i = 0; i < pid->num_upids; i++, upid++)
+  proc_flush_task_from_pid_ns(task, upid->pid_ns);
+}
+
 static struct dentry *proc_pid_instantiate(struct inode *dir,
        struct dentry * dentry,
        struct task_struct *task, const void *ptr)
Index: lx26-21-mm2/fs/proc/inode.c
===================================================================
--- lx26-21-mm2.orig/fs/proc/inode.c 2007-05-22 16:51:19.000000000 -0700
+++ lx26-21-mm2/fs/proc/inode.c 2007-05-22 16:59:56.000000000 -0700
@@ -6,6 +6,7 @@

 #include <linux/time.h>
 #include <linux/proc_fs.h>
+#include <linux/hardirq.h>
 #include <linux/kernel.h>
 #include <linux/mm.h>
 #include <linux/string.h>
@@ -74,8 +75,6 @@ static void proc_delete_inode(struct ino
  clear_inode(inode);
 }

-struct vfsmount *proc_mnt;
-
 static void proc_read_inode(struct inode * inode)
 {
  inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
@@ -458,6 +457,8 @@ out_mod:

 int proc_fill_super(struct super_block *s, void *data, int silent)
 {
+ struct pid_namespace *pid_ns = data;
+ struct proc_inode *ei;
  struct inode * root_inode;

  s->s_flags |= MS_NODIRATIME | MS_NOSUID | MS_NOEXEC;
@@ -466,6 +467,7 @@ int proc_fill_super(struct super_block *
  s->s_magic = PROC_SUPER_MAGIC;
  s->s_op = &proc_sops;
  s->s_time_gran = 1;
```

```
+ s->s_fs_info = pid_ns;

  de_get(&proc_root);
  root_inode = proc_get_inode(s, PROC_ROOT_INO, &proc_root);
@@ -476,6 +478,11 @@ int proc_fill_super(struct super_block *
  s->s_root = d_alloc_root(root_inode);
  if (!s->s_root)
   goto out_no_root;
+ /* Seed the root directory with a pid so it doesn't need
+  * to be special in base.c.
+  */
+ ei = PROC_I(root_inode);
+ ei->pid = find_get_pid(1);
  return 0;

 out_no_root:
Index: lx26-21-mm2/fs/proc/root.c
===================================================================
--- lx26-21-mm2.orig/fs/proc/root.c 2007-05-22 16:51:19.000000000 -0700
+++ lx26-21-mm2/fs/proc/root.c 2007-05-22 16:59:56.000000000 -0700
@@ -12,32 +12,56 @@
 #include <linux/time.h>
 #include <linux/proc_fs.h>
 #include <linux/stat.h>
+#include <linux/hardirq.h>
 #include <linux/init.h>
 #include <linux/sched.h>
 #include <linux/module.h>
 #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_sb(struct super_block *s, void *data)
+{
+ struct pid_namespace *pid_ns = data;
+ if (s->s_fs_info == pid_ns)
+  return 1;
+ return 0;
+}
+
 static int proc_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
 {
```

```
- 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);
+ int error;
+ struct super_block *s;
+ struct pid_namespace *pid_ns;
+
+ /*
+  * We can eventually derive this out of whatever mount
+  * arguments the user supplies, but just take it from
+  * current for now.
+  */
+ pid_ns = task_active_pid_ns(current);
+
+ s = sget(fs_type, proc_test_sb, set_anon_super, pid_ns);
+ if (IS_ERR(s))
+  return PTR_ERR(s);
+
+ error = proc_fill_super(s, pid_ns, 0);
+ if (error) {
+  deactivate_super(s);
+  return error;
  }
- return get_sb_single(fs_type, flags, data, proc_fill_super, mnt);
+
+ if (!pid_ns->proc_mnt)
+  pid_ns->proc_mnt = mnt;
+
+ do_remount_sb(s, flags, data, 0);
+ return simple_set_mnt(mnt, s);
 }

 static struct file_system_type proc_fs_type = {
@@ -54,12 +78,6 @@ void __init proc_root_init(void)
  err = register_filesystem(&proc_fs_type);
  if (err)
   return;
- proc_mnt = kern_mount(&proc_fs_type);
- 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);
Index: lx26-21-mm2/include/linux/pid_namespace.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:53.000000000 -0700
+++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 16:59:56.000000000 -0700
@@ -33,6 +33,7 @@ struct pid_namespace {
  int last_pid;
  struct task_struct *child_reaper;
  atomic_t terminating;
+ struct vfsmount *proc_mnt;
 };

 extern struct pid_namespace init_pid_ns;
Index: lx26-21-mm2/include/linux/proc_fs.h
===================================================================
--- lx26-21-mm2.orig/include/linux/proc_fs.h 2007-05-22 16:51:19.000000000 -0700
+++ lx26-21-mm2/include/linux/proc_fs.h 2007-05-22 16:59:56.000000000 -0700
@@ -125,7 +125,6 @@ extern struct proc_dir_entry *create_pro
       struct proc_dir_entry *parent);
 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);
 extern struct inode *proc_get_inode(struct super_block *, unsigned int, struct proc_dir_entry *);
```

Subject: [RFC][PATCH 15/16] Enable signaling child reaper from parent ns.
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:15:16 GMT
View Forum Message <> Reply to Message

Subject: Enable signaling child reaper from parent ns.

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

The reaper of a child namespace must receive signals from its parent pid
namespace but not receive any signals from its own namespace.

This is a very early draft :-) and following tests seem to pass

- Successfully kill child reaper from parent namespace (init_pid_ns)

- Fail to kill child reaper from within its namespace (non init_pid_ns)

- kill -1 1 from init_pid_ns seemed to work (rescanned inittab)

TODO:
- Test async io and SIGIO delivery.

- Allow any legitimate signals that the child reaper can receive
  from within its namespace? (we block all signals now)

      - Sending SIGKILL to the child reaper of a namespace terminates the
        namespace But if the namespace remounted /proc from user space,
  /proc would remain mounted even after reaper and other process in
  the namespace go away.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 kernel/signal.c | 22 ++++++++++++++++++++-
 1 file changed, 21 insertions(+), 1 deletion(-)

Index: lx26-21-mm2/kernel/signal.c
==========
--- lx26-21-mm2.orig/kernel/signal.c 2007-05-22 16:59:42.000000000 -0700
+++ lx26-21-mm2/kernel/signal.c 2007-05-22 16:59:57.000000000 -0700
@@ -507,6 +507,20 @@ static int check_kill_permission(int sig
     && !capable(CAP_KILL))
   return error;

+ /*
+ * If t is the reaper of its namespace and someone from that
+ * namespace is trying to send a signal.
+ *
+ * Note: If some one from parent namespace is sending a signal,
+ *   task_child_reaper() != t and we allow the signal.
+ *
+ * In the child namespace, does this block even legitimate signals
+ * like the ones telinit sends to /sbin/init ?
+ *
+ */
+ if ((!is_global_init(t)) && (t == task_child_reaper(t)))
+ return -EPERM;
+
  error = security_task_kill(t, info, sig, 0);
  if (!error)
   audit_signal_info(sig, t); /* Let audit system see the signal */
@@ -1910,7 +1924,13 @@ relock:

```
   /*
    * Init of a pid space gets no signals it doesn't want from
    * within that pid space. It can of course get signals from
-   * its parent pid space.
+   * its parent pid space. But we have no way of knowing the
+   * namespace from which the signal was sent. For now check
+   * if we are global init here and add additional checks in
+   * sys_kill() and friends.
+   *
+   * Note that t == task_child_reaper(t) implies t is the global
+   * init (and we are in init_pid_ns).
    */
   if (current == task_child_reaper(current))
    continue;
```

Subject: [RFC][PATCH 16/16] Move inline functions to sched.h
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 01:15:36 GMT
View Forum Message <> Reply to Message

Subject: Move inline functions to sched.h

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

task_active_pid_ns() and pid_to_nr() cannot both be inline since they
are currently defined in different files (pid_namespace.h and pid.h)
These files depend on each other and sched.h and to resolve the
circular dependency some functions have to be extern even if they all
should really be inline.

This patch moves all these functions to sched.h and makes them all inline.

TODO: Fold these changes into the appropriate patches in the patchset.
      Any concern with adding these to sched.h ?

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

```
 include/linux/pid.h          |   2
 include/linux/pid_namespace.h |   8 ---
 include/linux/sched.h        |  92 ++++++++++++++++++++++++++++++++++++++++++
 kernel/pid.c                 |  86 ----------------------------------------
 4 files changed, 92 insertions(+), 96 deletions(-)
```

```
Index: lx26-21-mm2/kernel/pid.c
===================================================================
--- lx26-21-mm2.orig/kernel/pid.c 2007-05-23 13:25:56.000000000 -0700
+++ lx26-21-mm2/kernel/pid.c 2007-05-23 13:25:56.000000000 -0700
@@ -246,49 +246,6 @@ static int init_upid(struct upid *upid,
  return 0;
 }

-static struct upid *pid_active_upid(struct pid *pid)
-{
- return &pid->upid_list[0];
-}
-
-/*
- * Return the active pid namespace of the process @pid.
- *
- * Note:
- * To avoid having to use an extra pointer in struct pid to keep track
- * of active pid namespace, dup_struct_pid() maintains the order of
- * entries in 'pid->upid_list' such that the youngest (or the 'active')
- * pid namespace is the first entry and oldest (init_pid_ns) is the last
- * entry in the list.
- */
-struct pid_namespace *pid_active_pid_ns(struct pid *pid)
-{
- return pid_active_upid(pid)->pid_ns;
-}
-EXPORT_SYMBOL_GPL(pid_active_pid_ns);
-
-/*
- * Return the parent pid_namespace of the active pid namespace of @tsk.
- *
- * Note:
- *  Refer to function header of pid_active_pid_ns() for information on
- *  the order of entries in pid->upid_list. Based on the order, the parent
- *  pid namespace of the active pid namespace of @tsk is just the second
- *  entry in the process's pid->upid_list.
- *
- *  Parent pid namespace of init_pid_ns is init_pid_ns itself.
- */
-static struct pid_namespace *task_active_pid_ns_parent(struct task_struct *tsk)
-{
- int idx = 0;
- struct pid *pid = task_pid(tsk);
-
- if (pid->num_upids > 1)
-  idx++;
```

```
-
- return pid->upid_list[idx].pid_ns;
-}
-
 /*
  * Return the child reaper of @tsk.
  *
@@ -325,49 +282,6 @@ struct task_struct *task_child_reaper(st
 }
 EXPORT_SYMBOL(task_child_reaper);

-/*
- * Return the pid_t by which the process @pid is known in the pid
- * namespace @ns.
- *
- * Return 0 if:
- *  - @pid is NULL (eg: procfs calls this for task_pgrp(init_task)
- *    which is NULL).
- *
- *  - process does not have pid_t in the namespace @ns (eg: parent
- *    process of a child reaper does not exist in the child namespace.
- *    A getppid() call by the child reaper results in 0).
- */
-pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid)
-{
- int i;
- struct upid *upid;
-
- if (!pid)
-  return 0;
-
- upid = &pid->upid_list[0];
- for (i = 0; i < pid->num_upids; i++, upid++) {
-  if (upid->pid_ns == ns)
-   return upid->nr;
- }
- return 0;
-}
-EXPORT_SYMBOL_GPL(pid_to_nr_in_ns);
-
-/*
- * Return the pid_t by which the process @pid is known in the active
- * pid namespace of the caller.
- *
- * pid_to_nr() cannot be static inline if task_active_pid_ns() is
- * inline as it would cause a circular dependency between pid.h
- * and pid_namespace.h.
- */
```

```
-pid_t pid_to_nr(struct pid *pid)
-{
- return pid_to_nr_in_ns(task_active_pid_ns(current), pid);
-}
-EXPORT_SYMBOL_GPL(pid_to_nr);
-
 #ifdef CONFIG_PID_NS
 static int init_ns_pidmap(struct pid_namespace *ns)
 {
Index: lx26-21-mm2/include/linux/sched.h
===================================================================
--- lx26-21-mm2.orig/include/linux/sched.h 2007-05-23 13:25:56.000000000 -0700
+++ lx26-21-mm2/include/linux/sched.h 2007-05-23 13:25:56.000000000 -0700
@@ -1176,6 +1176,98 @@ struct pid_namespace;
 extern int is_global_init(struct task_struct *tsk);
 extern int is_container_init(struct task_struct *tsk);

+/*
+ * Return the active upid of the process @pid.
+ *
+ * Note:
+ * To avoid having to use an extra pointer in struct pid to keep track
+ * of active pid namespace, dup_struct_pid() maintains the order of
+ * entries in 'pid->upid_list' such that the youngest (or the 'active')
+ * pid namespace is the first entry and oldest (init_pid_ns) is the last
+ * entry in the list.
+ */
+static inline struct upid *pid_active_upid(struct pid *pid)
+{
+ return &pid->upid_list[0];
+}
+
+static inline struct pid_namespace *pid_active_pid_ns(struct pid *pid)
+{
+ return pid_active_upid(pid)->pid_ns;
+}
+
+static inline struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
+{
+ return pid_active_pid_ns(task_pid(tsk));
+}
+
+/*
+ * Return the parent pid_namespace of the active pid namespace of @tsk.
+ *
+ * Note:
+ *  Refer to function header of pid_active_pid_ns() for information on
+ *  the order of entries in pid->upid_list. Based on the order, the parent
```

```
+ *  pid namespace of the active pid namespace of @tsk is just the second
+ *  entry in the process's pid->upid_list.
+ *
+ *  Parent pid namespace of init_pid_ns is init_pid_ns itself.
+ */
+static inline struct pid_namespace *task_active_pid_ns_parent(
+  struct task_struct *tsk)
+{
+ int idx = 0;
+ struct pid *pid = task_pid(tsk);
+
+ if (pid->num_upids > 1)
+  idx++;
+
+ return pid->upid_list[idx].pid_ns;
+}
+
+extern struct task_struct *task_child_reaper(struct task_struct *tsk);
+
+/*
+ * Return the pid_t by which the process @pid is known in the pid
+ * namespace @ns.
+ *
+ * Return 0 if:
+ *  - @pid is NULL (eg: procfs calls this for task_pgrp(init_task)
+ *    which is NULL).
+ *
+ *  - process does not have pid_t in the namespace @ns (eg: parent
+ *    process of a child reaper does not exist in the child namespace.
+ *    A getppid() call by the child reaper results in 0).
+ */
+static inline pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid)
+{
+ int i;
+ struct upid *upid;
+
+ WARN_ON(pid == 0);
+ if (!pid)
+  return 0;
+
+ upid = &pid->upid_list[0];
+ for (i = 0; i < pid->num_upids; i++, upid++) {
+  if (upid->pid_ns == ns)
+   return upid->nr;
+ }
+ return 0;
+}
+
```

```
+/*
+ * Return the pid_t by which the process @pid is known in the active
+ * pid namespace of the caller.
+ *
+ * pid_to_nr() cannot be static inline if task_active_pid_ns() is
+ * inline as it would cause a circular dependency between pid.h
+ * and pid_namespace.h.
+ */
+static inline pid_t pid_to_nr(struct pid *pid)
+{
+ return pid_to_nr_in_ns(task_active_pid_ns(current), pid);
+}
+
 extern struct pid *cad_pid;

 extern void free_task(struct task_struct *tsk);
Index: lx26-21-mm2/include/linux/pid.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid.h 2007-05-23 13:25:56.000000000 -0700
+++ lx26-21-mm2/include/linux/pid.h 2007-05-23 13:25:56.000000000 -0700
@@ -122,8 +122,6 @@ extern struct pid *dup_struct_pid(enum c
   unsigned long clone_flags, struct task_struct *new_task);
 extern void FASTCALL(free_pid(struct pid *pid));

-extern pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid);
-extern pid_t pid_to_nr(struct pid *pid);
 extern void zap_pid_ns_processes(struct pid_namespace *pid_ns);

 #define do_each_pid_task(pid, type, task)    \
Index: lx26-21-mm2/include/linux/pid_namespace.h
===================================================================
--- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-23 13:25:56.000000000 -0700
+++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-23 13:25:56.000000000 -0700
@@ -44,18 +44,10 @@ static inline void get_pid_ns(struct pid
 }

 extern void free_pid_ns(struct kref *kref);
-extern struct pid_namespace *pid_active_pid_ns(struct pid *pid);

 static inline void put_pid_ns(struct pid_namespace *ns)
 {
  kref_put(&ns->kref, free_pid_ns);
 }

-static inline struct pid_namespace *task_active_pid_ns(struct task_struct *tsk)
-{
- return pid_active_pid_ns(task_pid(tsk));
-}
```

-
-extern struct task_struct *task_child_reaper(struct task_struct *tsk);
-
 #endif /* _LINUX_PID_NS_H */

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

---

Subject: Re: [RFC][PATCH 04/16] Use pid_to_nr() in process info functions
Posted by xemul on Thu, 24 May 2007 08:22:25 GMT
View Forum Message <> Reply to Message

> ===
> --- lx26-21-mm2.orig/kernel/timer.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/kernel/timer.c 2007-05-22 16:59:44.000000000 -0700
> @@ -945,7 +945,7 @@ asmlinkage long sys_getppid(void)
> int pid;
>
> rcu_read_lock();
> - pid = rcu_dereference(current->real_parent)->tgid;
> + pid = pid_to_nr(task_parent_tgid(current));

This breaks fsys_getppid() call in ia64...

> rcu_read_unlock();
>
> return pid;

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by xemul on Thu, 24 May 2007 08:28:59 GMT
View Forum Message <> Reply to Message

> Index: lx26-21-mm2/kernel/pid.c
> ===
> --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:34.000000000 -0700
> +++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
> @@ -71,6 +71,27 @@ struct pid_namespace init_pid_ns = {
> .child_reaper = &init_task
> };
>

> +
> +/**
> + * is_global_init - check if a task structure is init
> + * @tsk: Task structure to be checked.
> + *
> + * Check if a task structure is the first user space task the kernel created.
> + */
> +int is_global_init(struct task_struct *tsk)
> +{
> + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);

This can OOPS if you pass arbitrary task to this call...
tsk->nsproxy can already be NULL.

> +}
> +
> +/*
> + * is_container_init:
> + * check whether in the task is init in it's own pid namespace.
> + */
> +int is_container_init(struct task_struct *tsk)
> +{
> + return tsk->pid == 1;
> +}
> +
> /*
>   * Note: disable interrupts while the pidmap_lock is held as an
>   * interrupt might come in and do read_lock(&tasklist_lock).
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
>

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by xemul on Thu, 24 May 2007 08:29:10 GMT
View Forum Message <> Reply to Message

> Index: lx26-21-mm2/kernel/pid.c
> ===
> --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:34.000000000 -0700
> +++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
> @@ -71,6 +71,27 @@ struct pid_namespace init_pid_ns = {

> .child_reaper = &init_task
> };
>
> +
> +/**
> + * is_global_init - check if a task structure is init
> + * @tsk: Task structure to be checked.
> + *
> + * Check if a task structure is the first user space task the kernel created.
> + */
> +int is_global_init(struct task_struct *tsk)
> +{
> + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);

This can OOPS if you pass arbitrary task to this call...
tsk->nsproxy can already be NULL.

> +}
> +
> +/*
> + * is_container_init:
> + * check whether in the task is init in it's own pid namespace.
> + */
> +int is_container_init(struct task_struct *tsk)
> +{
> + return tsk->pid == 1;
> +}
> +
> /*
>   * Note: disable interrupts while the pidmap_lock is held as an
>   * interrupt might come in and do read_lock(&tasklist_lock).
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
>

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

---

Subject: Re: [RFC][PATCH 07/16] Move alloc_pid call to copy_process
Posted by xemul on Thu, 24 May 2007 08:35:48 GMT
View Forum Message <> Reply to Message

> Index: lx26-21-mm2/kernel/pid.c
> ===

> --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
> +++ lx26-21-mm2/kernel/pid.c 2007-05-22 17:06:48.000000000 -0700
> @@ -216,6 +216,10 @@ fastcall void free_pid(struct pid *pid)
> /* We can be called with write_lock_irq(&tasklist_lock) held */
> unsigned long flags;
>
> + /* check this here to keep copy_process() cleaner */
> + if (unlikely(pid == &init_struct_pid))
> + return;
> +

This looks ugly to me.
That's the same as if we put
  if (ns == &init_pid_ns)
      return;
in put_pid_ns() call.

Such small struts of their own do not introduce any noticeable
effect, but when we have them in many places (and on fast patsh
like alloc_pid()) the performance hurts...

> spin_lock_irqsave(&pidmap_lock, flags);
> hlist_del_rcu(&pid->pid_chain);
> spin_unlock_irqrestore(&pidmap_lock, flags);
> @@ -224,12 +228,16 @@ fastcall void free_pid(struct pid *pid)
> call_rcu(&pid->rcu, delayed_put_pid);
> }
>
> -struct pid *alloc_pid(void)
> +struct pid *alloc_pid(enum copy_process_type copy_src)
> {
> struct pid *pid;
> enum pid_type type;
> int nr = -1;
>
> + /* check this here to keep copy_process() cleaner */
> + if (unlikely(copy_src == COPY_IDLE_PROCESS))
> + return &init_struct_pid;
> +
> pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
> if (!pid)
> goto out;
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
>

Subject: Re: [RFC][PATCH 08/16] Define/use pid->upid_list list.
Posted by xemul on Thu, 24 May 2007 08:57:33 GMT
View Forum Message <> Reply to Message

> +/*
> + * Return the pid_t by which the process @pid is known in the pid
> + * namespace @ns.
> + *
> + * Return 0 if:
> + * - @pid is NULL (eg: procfs calls this for task_pgrp(init_task)
> + *   which is NULL).
> + *
> + * - process does not have pid_t in the namespace @ns (eg: parent
> + *   process of a child reaper does not exist in the child namespace.
> + *   A getppid() call by the child reaper results in 0).
> + */
> +pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid)
> +{
> + int i;
> + struct upid *upid;
> +
> + if (!pid)
> + return 0;
> +
> + upid = &pid->upid_list[0];
> + for (i = 0; i < pid->num_upids; i++, upid++) {
> + if (upid->pid_ns == ns)
> +   return upid->nr;
> + }

This will make users of the kernel who do not need the
pid namespaces suffer from performance loss. Why not
introduce a CONFIG_PID_NS option?

> + return 0;
> +}
> +EXPORT_SYMBOL_GPL(pid_to_nr_in_ns);
> @@ -2182,6 +2185,8 @@ 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;
> + /* TODO get pid_ns from proc mnt rather than current */

> + struct pid_namespace *ns = task_active_pid_ns(current);

IMHO this is not a TODO, but MUSTDO.
When you have one /proc mount accessed from different pid
namespaces you may be in situation when one pidnr (like 123)
represents different tasks in these namespaces and thus the
inode staying behind the dentry with d_name "123" must
reference two tasks. When we scale the model to N namespace
inode must reference N tasks. But I don't see it in the
patches...

>
> if (!reaper)
> goto out_no_task;

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 09:10:52 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@sw.ru] wrote:
| > Index: lx26-21-mm2/kernel/pid.c
| > ===
| > --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:34.000000000 -0700
| > +++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
| > @@ -71,6 +71,27 @@ struct pid_namespace init_pid_ns = {
| > .child_reaper = &init_task
| > };
| >
| > +
| > +/**
| > + * is_global_init - check if a task structure is init
| > + * @tsk: Task structure to be checked.
| > + *
| > + * Check if a task structure is the first user space task the kernel created.
| > + */
| > +int is_global_init(struct task_struct *tsk)
| > +{
| > + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);
|
| This can OOPS if you pass arbitrary task to this call...
| tsk->nsproxy can already be NULL.

Hmm. You are right. btw, this could be a bisect issue. Patch 9 of uses

pid_ns from pid->upid_list and removes nsproxy->pid_ns.

|
| > +}
| > +
| > +/*
| > + * is_container_init:
| > + * check whether in the task is init in it's own pid namespace.
| > + */
| > +int is_container_init(struct task_struct *tsk)
| > +{
| > + return tsk->pid == 1;
| > +}
| > +
| > /*
| >   * Note: disable interrupts while the pidmap_lock is held as an
| >   * interrupt might come in and do read_lock(&tasklist_lock).
| > _____
| > Containers mailing list
| > Containers@lists.linux-foundation.org
| > https://lists.linux-foundation.org/mailman/listinfo/containers
| >
| > _____
| > Devel mailing list
| > Devel@openvz.org
| > https://openvz.org/mailman/listinfo/devel
| >

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

---

Subject: Re: [RFC][PATCH 07/16] Move alloc_pid call to copy_process
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 09:17:47 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@sw.ru] wrote:
| > Index: lx26-21-mm2/kernel/pid.c
| > ===
| > --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
| > +++ lx26-21-mm2/kernel/pid.c 2007-05-22 17:06:48.000000000 -0700
| > @@ -216,6 +216,10 @@ fastcall void free_pid(struct pid *pid)
| >   /* We can be called with write_lock_irq(&tasklist_lock) held */
| >   unsigned long flags;
| >
| > + /* check this here to keep copy_process() cleaner */
| > + if (unlikely(pid == &init_struct_pid))

| > + return;
| > +
|
| This looks ugly to me.

I agree about the ugly part :-) but we need to distinguish
between idle thread and normal thread at some point in do_fork().

| That's the same as if we put
| if (ns == &init_pid_ns)
| return;
| in put_pid_ns() call.
|
| Such small struts of their own do not introduce any noticeable
| effect, but when we have them in many places (and on fast patsh
| like alloc_pid()) the performance hurts...

I agree and we have been trying to keep the impact as low as possible.
|
|
| >   spin_lock_irqsave(&pidmap_lock, flags);
| >   hlist_del_rcu(&pid->pid_chain);
| >   spin_unlock_irqrestore(&pidmap_lock, flags);
| > @@ -224,12 +228,16 @@ fastcall void free_pid(struct pid *pid)
| >   call_rcu(&pid->rcu, delayed_put_pid);
| > }
| >
| > -struct pid *alloc_pid(void)
| > +struct pid *alloc_pid(enum copy_process_type copy_src)
| > {
| >   struct pid *pid;
| >   enum pid_type type;
| >   int nr = -1;
| >
| > + /* check this here to keep copy_process() cleaner */
| > + if (unlikely(copy_src == COPY_IDLE_PROCESS))
| > + return &init_struct_pid;
| > +
| >   pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
| >   if (!pid)
| >   goto out;
| > _____
| > Containers mailing list
| > Containers@lists.linux-foundation.org
| > https://lists.linux-foundation.org/mailman/listinfo/containers
| >
| > _____
| > Devel mailing list

| > Devel@openvz.org
| > https://openvz.org/mailman/listinfo/devel
| >

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

---

Subject: Re: [RFC][PATCH 14/16] Introduce proc_mnt for pid_ns
Posted by xemul on Thu, 24 May 2007 09:23:01 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Subject: Introduce proc_mnt for pid_ns
>
> From: Dave Hansen <hansendc@us.ibm.com>
>
> The following patch completes the removal of the global proc_mnt.
> It fetches the mnt on which to do dentry invalidations from the
> pid_namespace in which the task appears.
>
> For now, there is only one pid namespace in mainline so this is
> straightforward. In the -lxc tree we'll have to do something
> more complex. The proc_flush_task() code takes a task, and
> needs to be able to find the corresponding proc superblocks on
> which that tasks's /proc/<pid> directories could appear. We
> can tell in which pid namespaces a task appears, so I put a
> pointer from the pid namespace to the corresponding proc_mnt.
>
> /proc currently has some special code to make sure that the root
> directory gets set up correctly. It proc_mnt variable in order
> to find its way to the root inode.
>
> Signed-off-by: Dave Hansen <haveblue@us.ibm.com>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
>
> fs/proc/base.c | 32 ++++++++++++++++++++++++++-
> fs/proc/inode.c | 11 +++++++-
> fs/proc/root.c | 52 ++----------
> include/linux/pid_namespace.h | 1
> include/linux/proc_fs.h | 1
> 5 files changed, 75 insertions(+), 22 deletions(-)
>
> Index: lx26-21-mm2/fs/proc/base.c
> ===
> --- lx26-21-mm2.orig/fs/proc/base.c 2007-05-22 16:59:49.000000000 -0700

```
> +++ lx26-21-mm2/fs/proc/base.c 2007-05-22 16:59:56.000000000 -0700
> @@ -2005,9 +2005,11 @@ static const struct inode_operations pro
>  };
>
>  /**
> - * proc_flush_task -  Remove dcache entries for @task from the /proc dcache.
> + * proc_flush_task_from_pid_ns -  Remove dcache entries for @task
> + *       from the /proc dcache.
>   *
>   * @task: task that should be flushed.
> + * @pid_ns: pid_namespace in which that task appears
>   *
>   * Looks in the dcache for
>   * /proc/@pid
> @@ -2025,11 +2027,22 @@ 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_from_pid_ns(struct task_struct *task,
> +    struct pid_namespace* pid_ns)
>  {
>   struct dentry *dentry, *leader, *dir;
>   char buf[PROC_NUMBUF];
>   struct qstr name;
> + struct vfsmount *proc_mnt;
> +
> + WARN_ON(!pid_ns);
> + /*
> +  * It is possible that no /procs have been instantiated
> +  * for this particular pid namespace.
> +  */
> + if (!pid_ns->proc_mnt)
> +  return;
> + proc_mnt = pid_ns->proc_mnt;
>
>   name.name = buf;
>   name.len = snprintf(buf, sizeof(buf), "%d", task->pid);
> @@ -2071,6 +2084,21 @@ out:
>   return;
>  }
>
> +void proc_flush_task(struct task_struct *task)
> +{
> + int i;
> + struct pid *pid;
> + struct upid* upid;
> +
```

> + pid = task_pid(task);
> + if (!pid)
> + return;
> +
> + upid = &pid->upid_list[0];
> + for (i = 0; i < pid->num_upids; i++, upid++)
> + proc_flush_task_from_pid_ns(task, upid->pid_ns);
> +}
> +
> static struct dentry *proc_pid_instantiate(struct inode *dir,
> struct dentry * dentry,
> struct task_struct *task, const void *ptr)
> Index: lx26-21-mm2/fs/proc/inode.c
> ===
> --- lx26-21-mm2.orig/fs/proc/inode.c 2007-05-22 16:51:19.000000000 -0700
> +++ lx26-21-mm2/fs/proc/inode.c 2007-05-22 16:59:56.000000000 -0700
> @@ -6,6 +6,7 @@
>
> #include <linux/time.h>
> #include <linux/proc_fs.h>
> +#include <linux/hardirq.h>
> #include <linux/kernel.h>
> #include <linux/mm.h>
> #include <linux/string.h>
> @@ -74,8 +75,6 @@ static void proc_delete_inode(struct ino
> clear_inode(inode);
> }
>
> -struct vfsmount *proc_mnt;
> -
> static void proc_read_inode(struct inode * inode)
> {
> inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
> @@ -458,6 +457,8 @@ out_mod:
>
> int proc_fill_super(struct super_block *s, void *data, int silent)
> {
> + struct pid_namespace *pid_ns = data;
> + struct proc_inode *ei;
> struct inode * root_inode;
>
> s->s_flags |= MS_NODIRATIME | MS_NOSUID | MS_NOEXEC;
> @@ -466,6 +467,7 @@ int proc_fill_super(struct super_block *
> s->s_magic = PROC_SUPER_MAGIC;
> s->s_op = &proc_sops;
> s->s_time_gran = 1;
> + s->s_fs_info = pid_ns;

This must be done in set callback for sget. Otherwise we
may have two superblocks with the same pid_ns...

> de_get(&proc_root);
> root_inode = proc_get_inode(s, PROC_ROOT_INO, &proc_root);
> @@ -476,6 +478,11 @@ int proc_fill_super(struct super_block *
> s->s_root = d_alloc_root(root_inode);
> if (!s->s_root)
> goto out_no_root;
> + /* Seed the root directory with a pid so it doesn't need
> + * to be special in base.c.
> + */
> + ei = PROC_I(root_inode);
> + ei->pid = find_get_pid(1);
> return 0;
>
> out_no_root:
> Index: lx26-21-mm2/fs/proc/root.c
> ===
> --- lx26-21-mm2.orig/fs/proc/root.c 2007-05-22 16:51:19.000000000 -0700
> +++ lx26-21-mm2/fs/proc/root.c 2007-05-22 16:59:56.000000000 -0700
> @@ -12,32 +12,56 @@
> #include <linux/time.h>
> #include <linux/proc_fs.h>
> #include <linux/stat.h>
> +#include <linux/hardirq.h>
> #include <linux/init.h>
> #include <linux/sched.h>
> #include <linux/module.h>
> #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_sb(struct super_block *s, void *data)
> +{
> + struct pid_namespace *pid_ns = data;
> + if (s->s_fs_info == pid_ns)
> + return 1;
> + return 0;
> +}
> +
> static int proc_get_sb(struct file_system_type *fs_type,
> int flags, const char *dev_name, void *data, struct vfsmount *mnt)

```
>  {
> - 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);
> + int error;
> + struct super_block *s;
> + struct pid_namespace *pid_ns;
> +
> + /*
> +  * We can eventually derive this out of whatever mount
> +  * arguments the user supplies, but just take it from
> +  * current for now.
> +  */
> + pid_ns = task_active_pid_ns(current);
> +
> + s = sget(fs_type, proc_test_sb, set_anon_super, pid_ns);
> + if (IS_ERR(s))
> +  return PTR_ERR(s);
> +
> + error = proc_fill_super(s, pid_ns, 0);
> + if (error) {
> +  deactivate_super(s);
> +  return error;
>   }
> - return get_sb_single(fs_type, flags, data, proc_fill_super, mnt);
> +
> + if (!pid_ns->proc_mnt)
> +  pid_ns->proc_mnt = mnt;
> +
> + do_remount_sb(s, flags, data, 0);
> + return simple_set_mnt(mnt, s);
>  }
>
>  static struct file_system_type proc_fs_type = {
> @@ -54,12 +78,6 @@ void __init proc_root_init(void)
>   err = register_filesystem(&proc_fs_type);
>   if (err)
>    return;
> - proc_mnt = kern_mount(&proc_fs_type);
> - 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);
> Index: lx26-21-mm2/include/linux/pid_namespace.h
> ===
> --- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:53.000000000 -0700
> +++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 16:59:56.000000000 -0700
> @@ -33,6 +33,7 @@ struct pid_namespace {
>   int last_pid;
>   struct task_struct *child_reaper;
>   atomic_t terminating;
> + struct vfsmount *proc_mnt;
> };
>
> extern struct pid_namespace init_pid_ns;
> Index: lx26-21-mm2/include/linux/proc_fs.h
> ===
> --- lx26-21-mm2.orig/include/linux/proc_fs.h 2007-05-22 16:51:19.000000000 -0700
> +++ lx26-21-mm2/include/linux/proc_fs.h 2007-05-22 16:59:56.000000000 -0700
> @@ -125,7 +125,6 @@ extern struct proc_dir_entry *create_pro
>   struct proc_dir_entry *parent);
> 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);
> extern struct inode *proc_get_inode(struct super_block *, unsigned int, struct proc_dir_entry *);
>
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
>

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by xemul on Thu, 24 May 2007 09:24:54 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@sw.ru] wrote:
> | > Index: lx26-21-mm2/kernel/pid.c

> | > ===
> | > --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:34.000000000 -0700
> | > +++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
> | > @@ -71,6 +71,27 @@ struct pid_namespace init_pid_ns = {
> | > .child_reaper = &init_task
> | > };
> | >
> | > +
> | > +/**
> | > + * is_global_init - check if a task structure is init
> | > + * @tsk: Task structure to be checked.
> | > + *
> | > + * Check if a task structure is the first user space task the kernel created.
> | > + */
> | > +int is_global_init(struct task_struct *tsk)
> | > +{
> | > + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);
> |
> | This can OOPS if you pass arbitrary task to this call...
> | tsk->nsproxy can already be NULL.
>
> Hmm. You are right. btw, this could be a bisect issue. Patch 9 of uses
> pid_ns from pid->upid_list and removes nsproxy->pid_ns.

Yes, but that patch is not good either.
task_pid(tsk) may become NULL as well and this will oops.

> |
> | > +}
> | > +
> | > +/*
> | > + * is_container_init:
> | > + * check whether in the task is init in it's own pid namespace.
> | > + */
> | > +int is_container_init(struct task_struct *tsk)
> | > +{
> | > + return tsk->pid == 1;
> | > +}
> | > +
> | > /*
> | > * Note: disable interrupts while the pidmap_lock is held as an
> | > * interrupt might come in and do read_lock(&tasklist_lock).
> | > _____
> | > Containers mailing list
> | > Containers@lists.linux-foundation.org
> | > https://lists.linux-foundation.org/mailman/listinfo/containers
> | >
> | > _____

> | > Devel mailing list
> | > Devel@openvz.org
> | > https://openvz.org/mailman/listinfo/devel
> | >
>

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

---

Subject: Re: [RFC][PATCH 07/16] Move alloc_pid call to copy_process
Posted by xemul on Thu, 24 May 2007 09:30:09 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@sw.ru] wrote:
> | > Index: lx26-21-mm2/kernel/pid.c
> | > ===
> | > --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
> | > +++ lx26-21-mm2/kernel/pid.c 2007-05-22 17:06:48.000000000 -0700
> | > @@ -216,6 +216,10 @@ fastcall void free_pid(struct pid *pid)
> | > /* We can be called with write_lock_irq(&tasklist_lock) held */
> | > unsigned long flags;
> | >
> | > + /* check this here to keep copy_process() cleaner */
> | > + if (unlikely(pid == &init_struct_pid))
> | > + return;
> | > +
> |
> | This looks ugly to me.
>
> I agree about the ugly part :-) but we need to distinguish
> between idle thread and normal thread at some point in do_fork().

Why not keep this as it was - pass the pid from do_fork() or
do_fork_idle(). Why is that bad?

> | That's the same as if we put
> | if (ns == &init_pid_ns)
> | return;
> | in put_pid_ns() call.
> |
> | Such small struts of their own do not introduce any noticeable
> | effect, but when we have them in many places (and on fast patsh
> | like alloc_pid()) the performance hurts...
>

> I agree and we have been trying to keep the impact as low as possible.

In this patches - yes. But when we have many patches with sucn
"hooks" this becomes noticeable and hard to debug.

> |
> |
> | >   spin_lock_irqsave(&pidmap_lock, flags);
> | >   hlist_del_rcu(&pid->pid_chain);
> | >   spin_unlock_irqrestore(&pidmap_lock, flags);
> | > @@ -224,12 +228,16 @@ fastcall void free_pid(struct pid *pid)
> | >   call_rcu(&pid->rcu, delayed_put_pid);
> | > }
> | >
> | > -struct pid *alloc_pid(void)
> | > +struct pid *alloc_pid(enum copy_process_type copy_src)
> | > {
> | >   struct pid *pid;
> | >   enum pid_type type;
> | >   int nr = -1;
> | >
> | > + /* check this here to keep copy_process() cleaner */
> | > + if (unlikely(copy_src == COPY_IDLE_PROCESS))
> | > + return &init_struct_pid;
> | > +
> | >   pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
> | >   if (!pid)
> | >   goto out;
> | > _____
> | > Containers mailing list
> | > Containers@lists.linux-foundation.org
> | > https://lists.linux-foundation.org/mailman/listinfo/containers
> | >
> | > _____
> | > Devel mailing list
> | > Devel@openvz.org
> | > https://openvz.org/mailman/listinfo/devel
> | >
>

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

---

Subject: Re: [RFC][PATCH 08/16] Define/use pid->upid_list list.

Pavel Emelianov [xemul@sw.ru] wrote:
| > +/*
| > + * Return the pid_t by which the process @pid is known in the pid
| > + * namespace @ns.
| > + *
| > + * Return 0 if:
| > + * - @pid is NULL (eg: procfs calls this for task_pgrp(init_task)
| > + *   which is NULL).
| > + *
| > + * - process does not have pid_t in the namespace @ns (eg: parent
| > + *   process of a child reaper does not exist in the child namespace.
| > + *   A getppid() call by the child reaper results in 0).
| > + */
| > +pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid)
| > +{
| > + int i;
| > + struct upid *upid;
| > +
| > + if (!pid)
| > + return 0;
| > +
| > + upid = &pid->upid_list[0];
| > + for (i = 0; i < pid->num_upids; i++, upid++) {
| > + if (upid->pid_ns == ns)
| > +   return upid->nr;
| > + }
|
| This will make users of the kernel who do not need the
| pid namespaces suffer from performance loss. Why not
| introduce a CONFIG_PID_NS option?

CONFIG_PID_NS is in the next patch which actually enables cloning
pid namespace. But this code is common and we are trying to optimize.

Patch 16 makes this an inline function (plan to fold that diff into
this if there are no concerns with adding to sched.h).

Also, when number of pid namespaces is 1, the 'upid_list' is allocated
adjacent to the 'struct pid' (pls see alloc_struct_pid() for the case
num_upids == 1)

|
| > + return 0;
| > +}
| > +EXPORT_SYMBOL_GPL(pid_to_nr_in_ns);

| > @@ -2182,6 +2185,8 @@ 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;
| > + /* TODO get pid_ns from proc mnt rather than current */
| > + struct pid_namespace *ns = task_active_pid_ns(current);
|
| IMHO this is not a TODO, but MUSTDO.
| When you have one /proc mount accessed from different pid
| namespaces you may be in situation when one pidnr (like 123)
| represents different tasks in these namespaces and thus the
| inode staying behind the dentry with d_name "123" must
| reference two tasks. When we scale the model to N namespace
| inode must reference N tasks. But I don't see it in the
| patches...

We allow remounting /proc in pid namespace (pls see patches 13
and 14) and expect that /proc will be remounted in the new ns.

|
| >
| > if (!reaper)
| > goto out_no_task;

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

---

Subject: Re: [RFC][PATCH 0/16] Enable cloning of pid namespace
Posted by xemul on Thu, 24 May 2007 09:31:50 GMT
View Forum Message <> Reply to Message

I haven't found an appropriate patch, so I ask the
question here.

Consider we're sending a signal from parent namespace
to one of the child namespaces. What pid will be put
in siginfo as si_pid?

Pavel.

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

---

Subject: Re: [RFC][PATCH 07/16] Move alloc_pid call to copy_process
Posted by Sukadev Bhattiprolu on Thu, 24 May 2007 09:42:41 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@sw.ru] wrote:
| sukadev@us.ibm.com wrote:
| > Pavel Emelianov [xemul@sw.ru] wrote:
| > | > Index: lx26-21-mm2/kernel/pid.c
| > | > ===
| > | > --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
| > | > +++ lx26-21-mm2/kernel/pid.c 2007-05-22 17:06:48.000000000 -0700
| > | > @@ -216,6 +216,10 @@ fastcall void free_pid(struct pid *pid)
| > | > /* We can be called with write_lock_irq(&tasklist_lock) held */
| > | > unsigned long flags;
| > | >
| > | > + /* check this here to keep copy_process() cleaner */
| > | > + if (unlikely(pid == &init_struct_pid))
| > | > + return;
| > | > +
| > |
| > | This looks ugly to me.
| >
| > I agree about the ugly part :-) but we need to distinguish
| > between idle thread and normal thread at some point in do_fork().
|
| Why not keep this as it was - pass the pid from do_fork() or
| do_fork_idle(). Why is that bad?

When CLONE_NEWPID flag is specified and we create a new pid ns
we need the task_struct that will become the child reaper for
the new ns. The task_struct is allocated in copy_process().

We could make a second check for CLONE_NEWPID in copy_process()
and then initialize the reaper for the ns, but that could have
same or more performance impact ?

|
| > | That's the same as if we put
| > | if (ns == &init_pid_ns)
| > | return;
| > | in put_pid_ns() call.
| > |
| > | Such small struts of their own do not introduce any noticeable
| > | effect, but when we have them in many places (and on fast patsh
| > | like alloc_pid()) the performance hurts...
| >
| > I agree and we have been trying to keep the impact as low as possible.
|
| In this patches - yes. But when we have many patches with sucn

| "hooks" this becomes noticeable and hard to debug.
|
| > |
| > |
| > | > spin_lock_irqsave(&pidmap_lock, flags);
| > | > hlist_del_rcu(&pid->pid_chain);
| > | > spin_unlock_irqrestore(&pidmap_lock, flags);
| > | > @@ -224,12 +228,16 @@ fastcall void free_pid(struct pid *pid)
| > | > call_rcu(&pid->rcu, delayed_put_pid);
| > | > }
| > | >
| > | > -struct pid *alloc_pid(void)
| > | > +struct pid *alloc_pid(enum copy_process_type copy_src)
| > | > {
| > | > struct pid *pid;
| > | > enum pid_type type;
| > | > int nr = -1;
| > | >
| > | > + /* check this here to keep copy_process() cleaner */
| > | > + if (unlikely(copy_src == COPY_IDLE_PROCESS))
| > | > + return &init_struct_pid;
| > | > +
| > | > pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
| > | > if (!pid)
| > | > goto out;
| > | > _____
| > | > Containers mailing list
| > | > Containers@lists.linux-foundation.org
| > | > https://lists.linux-foundation.org/mailman/listinfo/containers
| > | >
| > | > _____
| > | > Devel mailing list
| > | > Devel@openvz.org
| > | > https://openvz.org/mailman/listinfo/devel
| > | >
| >

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

---

Subject: Re: [RFC][PATCH 14/16] Introduce proc_mnt for pid_ns
Posted by xemul on Thu, 24 May 2007 10:15:33 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Subject: Introduce proc_mnt for pid_ns

>
> From: Dave Hansen <hansendc@us.ibm.com>
>
> The following patch completes the removal of the global proc_mnt.
> It fetches the mnt on which to do dentry invalidations from the
> pid_namespace in which the task appears.
>
> For now, there is only one pid namespace in mainline so this is
> straightforward. In the -lxc tree we'll have to do something
> more complex. The proc_flush_task() code takes a task, and
> needs to be able to find the corresponding proc superblocks on
> which that tasks's /proc/<pid> directories could appear. We
> can tell in which pid namespaces a task appears, so I put a
> pointer from the pid namespace to the corresponding proc_mnt.
>
> /proc currently has some special code to make sure that the root
> directory gets set up correctly. It proc_mnt variable in order
> to find its way to the root inode.
>
> Signed-off-by: Dave Hansen <haveblue@us.ibm.com>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
>
> fs/proc/base.c | 32 ++++++++++++++++++++++++++-
> fs/proc/inode.c | 11 +++++++-
> fs/proc/root.c | 52 ++++++++++++++++++++++++++++++++++----------
> include/linux/pid_namespace.h | 1
> include/linux/proc_fs.h | 1
> 5 files changed, 75 insertions(+), 22 deletions(-)
>
> Index: lx26-21-mm2/fs/proc/base.c
> ===
> --- lx26-21-mm2.orig/fs/proc/base.c 2007-05-22 16:59:49.000000000 -0700
> +++ lx26-21-mm2/fs/proc/base.c 2007-05-22 16:59:56.000000000 -0700
> @@ -2005,9 +2005,11 @@ static const struct inode_operations pro
> };
>
> /**
> - * proc_flush_task - Remove dcache entries for @task from the /proc dcache.
> + * proc_flush_task_from_pid_ns - Remove dcache entries for @task
> +   from the /proc dcache.
>   *
>   * @task: task that should be flushed.
> + * @pid_ns: pid_namespace in which that task appears
>   *
>   * Looks in the dcache for
>   * /proc/@pid
> @@ -2025,11 +2027,22 @@ 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_from_pid_ns(struct task_struct *task,
> +    struct pid_namespace* pid_ns)
>  {
>   struct dentry *dentry, *leader, *dir;
>   char buf[PROC_NUMBUF];
>   struct qstr name;
> + struct vfsmount *proc_mnt;
> +
> + WARN_ON(!pid_ns);
> + /*
> +  * It is possible that no /procs have been instantiated
> +  * for this particular pid namespace.
> +  */
> + if (!pid_ns->proc_mnt)
> +  return;
> + proc_mnt = pid_ns->proc_mnt;
>
>   name.name = buf;
>   name.len = snprintf(buf, sizeof(buf), "%d", task->pid);
> @@ -2071,6 +2084,21 @@ out:
>   return;
>  }
>
> +void proc_flush_task(struct task_struct *task)
> +{
> + int i;
> + struct pid *pid;
> + struct upid* upid;
> +
> + pid = task_pid(task);
> + if (!pid)
> +  return;
> +
> + upid = &pid->upid_list[0];
> + for (i = 0; i < pid->num_upids; i++, upid++)
> +  proc_flush_task_from_pid_ns(task, upid->pid_ns);
> +}
> +
>  static struct dentry *proc_pid_instantiate(struct inode *dir,
>         struct dentry * dentry,
>         struct task_struct *task, const void *ptr)
> Index: lx26-21-mm2/fs/proc/inode.c
> ===================================================================
> --- lx26-21-mm2.orig/fs/proc/inode.c 2007-05-22 16:51:19.000000000 -0700
```

> +++ lx26-21-mm2/fs/proc/inode.c 2007-05-22 16:59:56.000000000 -0700
> @@ -6,6 +6,7 @@
>
> #include <linux/time.h>
> #include <linux/proc_fs.h>
> +#include <linux/hardirq.h>
> #include <linux/kernel.h>
> #include <linux/mm.h>
> #include <linux/string.h>
> @@ -74,8 +75,6 @@ static void proc_delete_inode(struct ino
>   clear_inode(inode);
> }
>
> -struct vfsmount *proc_mnt;
> -
> static void proc_read_inode(struct inode * inode)
> {
>   inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
> @@ -458,6 +457,8 @@ out_mod:
>
> int proc_fill_super(struct super_block *s, void *data, int silent)
> {
> + struct pid_namespace *pid_ns = data;
> + struct proc_inode *ei;
>   struct inode * root_inode;
>
>   s->s_flags |= MS_NODIRATIME | MS_NOSUID | MS_NOEXEC;
> @@ -466,6 +467,7 @@ int proc_fill_super(struct super_block *
>   s->s_magic = PROC_SUPER_MAGIC;
>   s->s_op = &proc_sops;
>   s->s_time_gran = 1;
> + s->s_fs_info = pid_ns;

One more thing I've just noticed - you don't get the namespace
here so after all the tasks die and namespace is freed we
have a proc mount pointing to freed namespace...

>   de_get(&proc_root);
>   root_inode = proc_get_inode(s, PROC_ROOT_INO, &proc_root);
> @@ -476,6 +478,11 @@ int proc_fill_super(struct super_block *
>   s->s_root = d_alloc_root(root_inode);
>   if (!s->s_root)
>   goto out_no_root;
> + /* Seed the root directory with a pid so it doesn't need
> + * to be special in base.c.
> + */
> + ei = PROC_I(root_inode);
> + ei->pid = find_get_pid(1);

>   return 0;
>
> out_no_root:
> Index: lx26-21-mm2/fs/proc/root.c
> ===
> --- lx26-21-mm2.orig/fs/proc/root.c 2007-05-22 16:51:19.000000000 -0700
> +++ lx26-21-mm2/fs/proc/root.c 2007-05-22 16:59:56.000000000 -0700
> @@ -12,32 +12,56 @@
> #include <linux/time.h>
> #include <linux/proc_fs.h>
> #include <linux/stat.h>
> +#include <linux/hardirq.h>
> #include <linux/init.h>
> #include <linux/sched.h>
> #include <linux/module.h>
> #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_sb(struct super_block *s, void *data)
> +{
> + struct pid_namespace *pid_ns = data;
> + if (s->s_fs_info == pid_ns)
> + return 1;
> + return 0;
> +}
> +
> static int proc_get_sb(struct file_system_type *fs_type,
>   int flags, const char *dev_name, void *data, struct vfsmount *mnt)
> {
> - 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);
> + int error;
> + struct super_block *s;
> + struct pid_namespace *pid_ns;

```
> +
> + /*
> +  * We can eventually derive this out of whatever mount
> +  * arguments the user supplies, but just take it from
> +  * current for now.
> +  */
> + pid_ns = task_active_pid_ns(current);
> +
> + s = sget(fs_type, proc_test_sb, set_anon_super, pid_ns);
> + if (IS_ERR(s))
> +  return PTR_ERR(s);
> +
> + error = proc_fill_super(s, pid_ns, 0);
> + if (error) {
> +  deactivate_super(s);
> +  return error;
>   }
> - return get_sb_single(fs_type, flags, data, proc_fill_super, mnt);
> +
> + if (!pid_ns->proc_mnt)
> +  pid_ns->proc_mnt = mnt;
> +
> + do_remount_sb(s, flags, data, 0);
> + return simple_set_mnt(mnt, s);
>  }
>
>  static struct file_system_type proc_fs_type = {
> @@ -54,12 +78,6 @@ void __init proc_root_init(void)
>   err = register_filesystem(&proc_fs_type);
>   if (err)
>    return;
> - proc_mnt = kern_mount(&proc_fs_type);
> - 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);
> Index: lx26-21-mm2/include/linux/pid_namespace.h
> ===================================================================
> --- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:53.000000000 -0700
> +++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 16:59:56.000000000 -0700
> @@ -33,6 +33,7 @@ struct pid_namespace {
>   int last_pid;
>   struct task_struct *child_reaper;
>   atomic_t terminating;
```

> + struct vfsmount *proc_mnt;
> };
>
> extern struct pid_namespace init_pid_ns;
> Index: lx26-21-mm2/include/linux/proc_fs.h
> ===
> --- lx26-21-mm2.orig/include/linux/proc_fs.h 2007-05-22 16:51:19.000000000 -0700
> +++ lx26-21-mm2/include/linux/proc_fs.h 2007-05-22 16:59:56.000000000 -0700
> @@ -125,7 +125,6 @@ extern struct proc_dir_entry *create_pro
> struct proc_dir_entry *parent);
> 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);
> extern struct inode *proc_get_inode(struct super_block *, unsigned int, struct proc_dir_entry *);
>
> _____
> Containers mailing list
> Containers@lists.linux-foundation.org
> https://lists.linux-foundation.org/mailman/listinfo/containers
>

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

---

Subject: Re: [RFC][PATCH 0/16] Enable cloning of pid namespace
Posted by Cedric Le Goater on Thu, 24 May 2007 10:19:46 GMT
View Forum Message <> Reply to Message

Pavel Emelianov wrote:
> I haven't found an appropriate patch, so I ask the
> question here.
>
> Consider we're sending a signal from parent namespace
> to one of the child namespaces. What pid will be put
> in siginfo as si_pid?

Would 0 be an issue ? it seems reasonable to pretend
that the signal was sent from the kernel and also set
si_code to SI_KERNEL.

C.

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

Subject: Re: [RFC][PATCH 11/16] Enable cloning pid namespace
Posted by serue on Thu, 24 May 2007 14:59:32 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
>
> Subject: Enable cloning pid namespace
>
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>
>
> When clone() is invoked with CLONE_NEWPID, create a new pid namespace
> and then create a new struct pid for the new process. Allocate pid_t's
> for the new process in the new pid namespace and all ancestor pid
> namespaces. Make the newly cloned process the session and process group
> leader.
>
> Since the active pid namespace is special and expected to be the first
> entry in pid->upid_list, preserve the order of pid namespaces when
> cloning without CLONE_NEWPID.
>
> TODO (partial list:)
>
> - Identify clone flags that should not be specified with CLONE_NEWPID
>   and return -EINVAL from copy_process(), if they are specified. (eg:
>   CLONE_THREAD|CLONE_NEWPID ?)
> - Add a privilege check for CLONE_NEWPID
>
> Changelog:
>
> 2.6.21-mm2-pidns3:
>   - 'struct upid' used to be called 'struct pid_nr' and a list of these
>   were hanging off of 'struct pid'. So, we renamed 'struct pid_nr'
>   and now hold them in a statically sized array in 'struct pid' since
>   the number of 'struct upid's for a process is known at process-
>   creation time
>
> 2.6.21-mm2:
> - [Serge Hallyn] Terminate other processes in pid ns when reaper is
>   exiting.
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> include/linux/pid.h | 3
> include/linux/pid_namespace.h | 5 -
> init/Kconfig | 9 +
> kernel/exit.c | 14 ++
> kernel/fork.c | 17 ++-
> kernel/pid.c | 209 ++----
> 6 files changed, 227 insertions(+), 30 deletions(-)

>
> Index: lx26-21-mm2/kernel/pid.c
> ===
> --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:50.000000000 -0700
> +++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:52.000000000 -0700
> @@ -32,6 +32,7 @@
> #define pid_hashfn(nr) hash_long((unsigned long)nr, pidhash_shift)
> static struct hlist_head *pid_hash;
> static int pidhash_shift;
> +static struct kmem_cache *pid1_cachep;
> static struct kmem_cache *pid_cachep;
> struct upid init_struct_upid = INIT_STRUCT_UPID;
> struct pid init_struct_pid = INIT_STRUCT_PID;
> @@ -250,7 +251,12 @@ static struct upid *pid_active_upid(stru
> /*
>   * Return the active pid namespace of the process @pid.
>   *
> - * Note: At present, there is only one pid namespace (init_pid_ns).
> + * Note:
> + * To avoid having to use an extra pointer in struct pid to keep track
> + * of active pid namespace, dup_struct_pid() maintains the order of
> + * entries in 'pid->upid_list' such that the youngest (or the 'active')
> + * pid namespace is the first entry and oldest (init_pid_ns) is the last
> + * entry in the list.
>   */
> struct pid_namespace *pid_active_pid_ns(struct pid *pid)
> {
> @@ -259,6 +265,64 @@ struct pid_namespace *pid_active_pid_ns(
> EXPORT_SYMBOL_GPL(pid_active_pid_ns);
>
> /*
> + * Return the parent pid_namespace of the active pid namespace of @tsk.
> + *
> + * Note:
> + * Refer to function header of pid_active_pid_ns() for information on
> + * the order of entries in pid->upid_list. Based on the order, the parent
> + * pid namespace of the active pid namespace of @tsk is just the second
> + * entry in the process's pid->upid_list.
> + *
> + * Parent pid namespace of init_pid_ns is init_pid_ns itself.
> + */
> +static struct pid_namespace *task_active_pid_ns_parent(struct task_struct *tsk)
> +{
> + int idx = 0;
> + struct pid *pid = task_pid(tsk);
> +
> + if (pid->num_upids > 1)
> + idx++;

> +
> + return pid->upid_list[idx].pid_ns;
> +}
> +
> +/*
> + * Return the child reaper of @tsk.
> + *
> + * Normally the child reaper of @tsk is simply the child reaper
> + * the active pid namespace of @tsk.
> + *
> + * But if @tsk is itself child reaper of a namespace, NS1, its child
> + * reaper depends on the caller. If someone from an ancestor namespace
> + * or, if the reaper himself is asking, return the reaper of our parent
> + * namespace.
> + *
> + * If someone from namespace NS1 (other than reaper himself) is asking,
> + * return reaper of NS1.
> + */
> +struct task_struct *task_child_reaper(struct task_struct *tsk)
> +{
> + struct pid_namespace *tsk_ns = task_active_pid_ns(tsk);
> + struct task_struct *tsk_reaper = tsk_ns->child_reaper;
> + struct pid_namespace *my_ns;
> +
> + /*
> + * TODO: Check if we need a lock here. ns->child_reaper
> + *   can change in do_exit() when reaper is exiting.
> + */
> +
> + if (tsk != tsk_reaper)
> + return tsk_reaper;
> +
> + my_ns = task_active_pid_ns(current);
> + if (my_ns != tsk_ns || current == tsk)
> + return task_active_pid_ns_parent(tsk)->child_reaper;

This is bogus. This value is never returned to userspace. It is always
used to make kernel decisions like forget_original_parent() and
signaling. As such, this unnecessarily slows down this function, and
has the potential of creating a very subtle bug down the line (if there
isn't one already).

A task has one reaper, period, and a fn called task_child_reaper()
should return that reaper, period.

Then if userspace ever wants to see that value (right now it doesn't),
then whoever calls task_child_reaper from inside NS1 on NS1's reaper can
send back 0.

-serge

```
> + return tsk_reaper;
> +}
> +EXPORT_SYMBOL(task_child_reaper);
> +
> +/*
>   * Return the pid_t by which the process @pid is known in the pid
>   * namespace @ns.
>   *
> @@ -301,15 +365,78 @@ pid_t pid_to_nr(struct pid *pid)
>  }
>  EXPORT_SYMBOL_GPL(pid_to_nr);
>
> +#ifdef CONFIG_PID_NS
> +static int init_ns_pidmap(struct pid_namespace *ns)
> +{
> + int i;
> +
> + atomic_set(&ns->pidmap[0].nr_free, BITS_PER_PAGE - 1);
> +
> + ns->pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
> + if (!ns->pidmap[0].page)
> +  return -ENOMEM;
> +
> + set_bit(0, ns->pidmap[0].page);
> +
> + for (i = 1; i < PIDMAP_ENTRIES; i++) {
> +  atomic_set(&ns->pidmap[i].nr_free, BITS_PER_PAGE);
> +  ns->pidmap[i].page = NULL;
> + }
> + return 0;
> +}
> +
> +static struct pid_namespace *alloc_pid_ns(void)
> +{
> + struct pid_namespace *ns;
> + int rc;
> +
> + ns = kzalloc(sizeof(struct pid_namespace), GFP_KERNEL);
> + if (!ns)
> +  return NULL;
> +
> + rc = init_ns_pidmap(ns);
> + if(rc) {
> +  kfree(ns);
> +  return NULL;
```

```
> + }
> +
> + kref_init(&ns->kref);
> +
> + return ns;
> +}
> +
> +#else
> +
> +static int alloc_pid_ns()
> +{
> + static int warned;
> +
> + if (!warned) {
> +  printk(KERN_INFO "WARNING: CLONE_NEWPID disabled\n");
> +  warned = 1;
> + }
> + return 0;
> +}
> +#endif  /*CONFIG_PID_NS*/
> +
> +void toss_pid(struct pid *pid)
> +{
> + if (pid->num_upids == 1)
> +  kmem_cache_free(pid1_cachep, pid);
> + else {
> +  kfree(pid->upid_list);
> +  kmem_cache_free(pid_cachep, pid);
> + }
> +}
> +
>  fastcall void put_pid(struct pid *pid)
>  {
>   if (!pid)
>    return;
>
>   if ((atomic_read(&pid->count) == 1) ||
> -      atomic_dec_and_test(&pid->count)) {
> -  kmem_cache_free(pid_cachep, pid);
> - }
> +      atomic_dec_and_test(&pid->count))
> +  toss_pid(pid);
>  }
>  EXPORT_SYMBOL_GPL(put_pid);
>
> @@ -345,15 +472,28 @@ static struct pid *alloc_struct_pid(int
>   enum pid_type type;
>   struct upid *upid_list;
```

>   void *pid_end;
> + struct kmem_cache *cachep = pid1_cachep;
>
> - /* for now we only support one pid namespace */
> - BUG_ON(num_upids != 1);
> - pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
> + if (num_upids > 1)
> + cachep = pid_cachep;
> +
> + pid = kmem_cache_alloc(cachep, GFP_KERNEL);
>   if (!pid)
>   return NULL;
>
> - pid_end = (void *)pid + sizeof(struct pid);
> - pid->upid_list = (struct upid *)pid_end;
> + if (num_upids == 1) {
> + pid_end = (void *)pid + sizeof(struct pid);
> + pid->upid_list = (struct upid *)pid_end;
> + } else {
> + int upid_list_size = num_upids * sizeof(struct upid);
> +
> + upid_list = kzalloc(upid_list_size, GFP_KERNEL);
> + if (!upid_list) {
> +   kmem_cache_free(pid_cachep, pid);
> +   return NULL;
> + }
> + pid->upid_list = upid_list;
> + }

I would much rather see the upid_list be a part of the struct pid, as in

```
 struct pid {
  ...
  struct upid upid_list[0];
 };
```

and the allocation done all at once using kmalloc.

If we really want to use a cache later, we could either use a cache only
if num_upids==1, or use a set of caches, creating a new cache every
time someone does clone(CLONE_NEWPID) to a new depth of num_upids.

Others may disagree with this, I realize my preference somewhat
subjective.

>
>   atomic_set(&pid->count, 1);
>   pid->num_upids = num_upids;

```
> @@ -364,7 +504,8 @@ static struct pid *alloc_struct_pid(int
>   return pid;
>  }
>
> -struct pid *dup_struct_pid(enum copy_process_type copy_src)
> +struct pid *dup_struct_pid(enum copy_process_type copy_src,
> +  unsigned long clone_flags, struct task_struct *new_task)
>  {
>   int rc;
>   int i;
> @@ -379,20 +520,38 @@ struct pid *dup_struct_pid(enum copy_pro
>    return &init_struct_pid;
>
>   num_upids = parent_pid->num_upids;
> + if (clone_flags & CLONE_NEWPID)
> +  num_upids++;
>
>   pid = alloc_struct_pid(num_upids);
>   if (!pid)
>    return NULL;
>
>   upid = &pid->upid_list[0];
> +
> + if (clone_flags & CLONE_NEWPID) {
> +  struct pid_namespace *new_pid_ns = alloc_pid_ns();
> +
> +  if (!new_pid_ns)
> +   goto out_free_pid;
> +
> +  new_pid_ns->child_reaper = new_task;
> +  rc = init_upid(upid, pid, new_pid_ns);
> +  if (rc < 0)
> +   goto out_free_pid;
> +  upid++;
> + }
> +
>   parent_upid = &parent_pid->upid_list[0];
>
> - for (i = 0; i < num_upids; i++, upid++, parent_upid++) {
> + for (i = 0; i < parent_pid->num_upids; i++, upid++, parent_upid++) {
>    rc = init_upid(upid, pid, parent_upid->pid_ns);
>    if (rc < 0)
>     goto out_free_pid;
>   }
>
> + new_task->pid = pid_active_upid(pid)->nr;
> +
>   return pid;
```

>
> out_free_pid:
> @@ -533,9 +692,21 @@ EXPORT_SYMBOL_GPL(find_get_pid);
>
> void free_pid_ns(struct kref *kref)
> {
> + int i;
> + int nr_free;
>   struct pid_namespace *ns;
>
>   ns = container_of(kref, struct pid_namespace, kref);
> +
> + BUG_ON(ns == &init_pid_ns);
> +
> + for (i = 0; i < PIDMAP_ENTRIES; i++) {
> + nr_free = atomic_read(&ns->pidmap[i].nr_free);
> + BUG_ON(nr_free != BITS_PER_PAGE);
> +
> + if (ns->pidmap[i].page)
> +   kfree(ns->pidmap[i].page);
> + }
>   kfree(ns);
> }
>
> @@ -566,14 +737,22 @@ void __init pidhash_init(void)
>
> void __init pidmap_init(void)
> {
> - int pid_elem_size;
> + int pid1_elem_size;
>
>   init_pid_ns.pidmap[0].page = kzalloc(PAGE_SIZE, GFP_KERNEL);
>   /* Reserve PID 0. We never call free_pidmap(0) */
>   set_bit(0, init_pid_ns.pidmap[0].page);
>   atomic_dec(&init_pid_ns.pidmap[0].nr_free);
>
> - pid_elem_size = sizeof(struct pid) + sizeof(struct upid);
> - pid_cachep = kmem_cache_create("pid+1upid", pid1_elem_size, 0,
> - SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
> + /*
> + * Cache for struct pids with more than one pid namespace
> + */
> + pid_cachep = KMEM_CACHE(pid, SLAB_PANIC);
> +
> + /*
> + * Cache for struct pids with exactly one pid namespace
> + */
> + pid1_elem_size = sizeof(struct pid) + sizeof(struct upid);

> + pid1_cachep = kmem_cache_create("pid+1upid", pid1_elem_size, 0,
> +   SLAB_HWCACHE_ALIGN|SLAB_PANIC, NULL, NULL);
> }
> Index: lx26-21-mm2/include/linux/pid.h
> ==========
> --- lx26-21-mm2.orig/include/linux/pid.h 2007-05-22 16:59:49.000000000 -0700
> +++ lx26-21-mm2/include/linux/pid.h 2007-05-22 16:59:52.000000000 -0700
> @@ -118,7 +118,8 @@ extern struct pid *FASTCALL(find_pid(int
> extern struct pid *find_get_pid(int nr);
> extern struct pid *find_ge_pid(int nr);
>
> -extern struct pid *dup_struct_pid(enum copy_process_type);
> +extern struct pid *dup_struct_pid(enum copy_process_type,
> + unsigned long clone_flags, struct task_struct *new_task);
> extern void FASTCALL(free_pid(struct pid *pid));
>
> extern pid_t pid_to_nr_in_ns(struct pid_namespace *ns, struct pid *pid);
> Index: lx26-21-mm2/include/linux/pid_namespace.h
> ==========
> --- lx26-21-mm2.orig/include/linux/pid_namespace.h 2007-05-22 16:59:50.000000000 -0700
> +++ lx26-21-mm2/include/linux/pid_namespace.h 2007-05-22 16:59:52.000000000 -0700
> @@ -54,9 +54,6 @@ static inline struct pid_namespace *task
>   return pid_active_pid_ns(task_pid(tsk));
> }
>
> -static inline struct task_struct *task_child_reaper(struct task_struct *tsk)
> -{
> - return task_active_pid_ns(tsk)->child_reaper;
> -}
> +extern struct task_struct *task_child_reaper(struct task_struct *tsk);
>
> #endif /* _LINUX_PID_NS_H */
> Index: lx26-21-mm2/init/Kconfig
> ==========
> --- lx26-21-mm2.orig/init/Kconfig 2007-05-22 16:58:36.000000000 -0700
> +++ lx26-21-mm2/init/Kconfig 2007-05-22 16:59:52.000000000 -0700
> @@ -250,6 +250,15 @@ config UTS_NS
>   vservers, to use uts namespaces to provide different
>   uts info for different servers. If unsure, say N.
>
> +config PID_NS
> + depends on EXPERIMENTAL
> + bool "PID Namespaces"
> + default n
> + help
> +   Support multiple PID namespaces. This allows containers, i.e.
> +   vservers to use separate different PID namespaces to different
> +   servers. If unsuare, say N.

> +
> config AUDIT
>   bool "Auditing support"
>   depends on NET
> Index: lx26-21-mm2/kernel/exit.c
> ===
> --- lx26-21-mm2.orig/kernel/exit.c 2007-05-22 16:59:46.000000000 -0700
> +++ lx26-21-mm2/kernel/exit.c 2007-05-22 16:59:52.000000000 -0700
> @@ -866,6 +866,7 @@ fastcall NORET_TYPE void do_exit(long co
> {
>   struct task_struct *tsk = current;
>   int group_dead;
> + struct pid_namespace *pid_ns = task_active_pid_ns(tsk);
>
>   profile_task_exit(tsk);
>
> @@ -875,10 +876,15 @@ 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 == 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;
> +
> + /*
> + * Note that we cannot use task_child_reaper() here because
> + * it returns reaper for parent pid namespace if tsk is itself
> + * the reaper of the active pid namespace.
> + */
> + if (unlikely(tsk == pid_ns->child_reaper)) {
> + if (pid_ns != &init_pid_ns)
> +   pid_ns->child_reaper = init_pid_ns.child_reaper;
>   else
>   panic("Attempted to kill init!");
>   }
> Index: lx26-21-mm2/kernel/fork.c
> ===
> --- lx26-21-mm2.orig/kernel/fork.c 2007-05-22 16:59:49.000000000 -0700
> +++ lx26-21-mm2/kernel/fork.c 2007-05-22 16:59:52.000000000 -0700
> @@ -1026,14 +1026,13 @@ static struct task_struct *copy_process(
>   if (p->binfmt && !try_module_get(p->binfmt->module))
>   goto bad_fork_cleanup_put_domain;
>
> - pid = dup_struct_pid(copy_src);
> + pid = dup_struct_pid(copy_src, clone_flags, p);
>   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_to_nr(pid);
>
> INIT_LIST_HEAD(&p->children);
> INIT_LIST_HEAD(&p->sibling);
> @@ -1255,11 +1254,17 @@ static struct task_struct *copy_process(
> __ptrace_link(p, current->parent);
>
> if (thread_group_leader(p)) {
> + struct pid *pgrp = task_pgrp(current);
> + struct pid *session = task_session(current);
> +
> + if (clone_flags & CLONE_NEWPID)
> + pgrp = session = pid;
> +
> p->signal->tty = current->signal->tty;
> - p->signal->pgrp = process_group(current);
> - set_signal_session(p->signal, process_session(current));
> - attach_pid(p, PIDTYPE_PGID, task_pgrp(current));
> - attach_pid(p, PIDTYPE_SID, task_session(current));
> + p->signal->pgrp = pid_to_nr(pgrp);
> + set_signal_session(p->signal, pid_to_nr(session));
> + attach_pid(p, PIDTYPE_PGID, pgrp);
> + attach_pid(p, PIDTYPE_SID, session);
>
> list_add_tail_rcu(&p->tasks, &init_task.tasks);
> __get_cpu_var(process_counts)++;

_____

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by serue on Thu, 24 May 2007 15:27:43 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
>
> Subject: Define is_global_init()
>
> From: Serge E. Hallyn <serue@us.ibm.com>
>
>
> is_init() is an ambiguous name for the pid==1 check. Split it into

> is_global_init() and is_container_init().
>
> A container init has it's tsk->pid == 1.
>
> A global init also has it's tsk->pid == 1, and it's active pid namespace
> is the init_pid_ns.
>
> Changelog:
>
> 2.6.21-mm2-pidns2:
>
> - [Sukadev Bhattiprolu] Changed is_container_init() calls in {powerpc,
>   ppc,avr32}/traps.c for the _exception() call to is_global_init().
>   This way, we kill only the container if the container's init has a
>   bug rather than force a kernel panic.
>
> Signed-off-by: Serge E. Hallyn <serue@us.ibm.com>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> arch/alpha/mm/fault.c | 2 +-
> arch/arm/mm/fault.c | 2 +-
> arch/arm26/mm/fault.c | 2 +-
> arch/avr32/kernel/traps.c | 2 +-
> arch/avr32/mm/fault.c | 6 +++---
> arch/i386/lib/usercopy.c | 2 +-
> arch/i386/mm/fault.c | 2 +-
> arch/ia64/mm/fault.c | 2 +-
> arch/m32r/mm/fault.c | 2 +-
> arch/m68k/mm/fault.c | 2 +-
> arch/mips/mm/fault.c | 2 +-
> arch/powerpc/kernel/traps.c | 2 +-
> arch/powerpc/mm/fault.c | 2 +-
> arch/powerpc/platforms/pseries/ras.c | 2 +-
> arch/ppc/kernel/traps.c | 2 +-
> arch/ppc/mm/fault.c | 2 +-
> arch/s390/lib/uaccess_pt.c | 2 +-
> arch/s390/mm/fault.c | 2 +-
> arch/sh/mm/fault.c | 2 +-
> arch/sh64/mm/fault.c | 6 +++---
> arch/um/kernel/trap.c | 2 +-
> arch/x86_64/mm/fault.c | 4 ++--
> arch/xtensa/mm/fault.c | 2 +-
> drivers/char/sysrq.c | 2 +-
> include/linux/sched.h | 13 +++++----------
> kernel/capability.c | 3 ++-
> kernel/exit.c | 2 +-
> kernel/kexec.c | 2 +-
> kernel/pid.c | 21 ++++++++++++++++++++

> kernel/sysctl.c | 2 +-
> mm/oom_kill.c | 4 ++--
> security/commoncap.c | 2 +-
> 32 files changed, 61 insertions(+), 46 deletions(-)
>
> Index: lx26-21-mm2/arch/alpha/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/alpha/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/alpha/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -192,7 +192,7 @@ do_page_fault(unsigned long address, uns
>   /* We ran out of memory, or some other thing happened to us that
>   made us unable to handle the page fault gracefully. */
>   out_of_memory:
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>   yield();
>   down_read(&mm->mmap_sem);
>   goto survive;
> Index: lx26-21-mm2/arch/arm/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/arm/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/arm/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -197,7 +197,7 @@ survive:
>   return fault;
>   }
>
> - if (!is_init(tsk))
> + if (!is_global_init(tsk))
>   goto out;
>
>   /*
> Index: lx26-21-mm2/arch/arm26/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/arm26/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/arm26/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -185,7 +185,7 @@ survive:
>   }
>
>   fault = -3; /* out of memory */
> - if (!is_init(tsk))
> + if (!is_global_init(tsk))
>   goto out;
>
>   /*
> Index: lx26-21-mm2/arch/i386/lib/usercopy.c
> ===
> --- lx26-21-mm2.orig/arch/i386/lib/usercopy.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/i386/lib/usercopy.c 2007-05-22 16:59:46.000000000 -0700

```
> @@ -748,7 +748,7 @@ survive:
>     retval = get_user_pages(current, current->mm,
>       (unsigned long )to, 1, 1, 0, &pg, NULL);
>
> -   if (retval == -ENOMEM && is_init(current)) {
> +   if (retval == -ENOMEM && is_global_init(current)) {
>      up_read(&current->mm->mmap_sem);
>      congestion_wait(WRITE, HZ/50);
>      goto survive;
> Index: lx26-21-mm2/arch/i386/mm/fault.c
> ===================================================================
> --- lx26-21-mm2.orig/arch/i386/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/i386/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -576,7 +576,7 @@ no_context:
>   */
>  out_of_memory:
>   up_read(&mm->mmap_sem);
> - if (is_init(tsk)) {
> + if (is_global_init(tsk)) {
>    yield();
>    down_read(&mm->mmap_sem);
>    goto survive;
> Index: lx26-21-mm2/arch/ia64/mm/fault.c
> ===================================================================
> --- lx26-21-mm2.orig/arch/ia64/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/ia64/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -279,7 +279,7 @@ ia64_do_page_fault (unsigned long addres
>
>    out_of_memory:
>   up_read(&mm->mmap_sem);
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>    yield();
>    down_read(&mm->mmap_sem);
>    goto survive;
> Index: lx26-21-mm2/arch/m32r/mm/fault.c
> ===================================================================
> --- lx26-21-mm2.orig/arch/m32r/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/m32r/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -272,7 +272,7 @@ no_context:
>   */
>  out_of_memory:
>   up_read(&mm->mmap_sem);
> - if (is_init(tsk)) {
> + if (is_global_init(tsk)) {
>    yield();
>    down_read(&mm->mmap_sem);
>    goto survive;
```

> Index: lx26-21-mm2/arch/m68k/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/m68k/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/m68k/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -181,7 +181,7 @@ good_area:
> */
> out_of_memory:
> up_read(&mm->mmap_sem);
> - if (is_init(current)) {
> + if (is_global_init(current)) {
> yield();
> down_read(&mm->mmap_sem);
> goto survive;
> Index: lx26-21-mm2/arch/mips/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/mips/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/mips/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -174,7 +174,7 @@ no_context:
> */
> out_of_memory:
> up_read(&mm->mmap_sem);
> - if (is_init(tsk)) {
> + if (is_global_init(tsk)) {
> yield();
> down_read(&mm->mmap_sem);
> goto survive;
> Index: lx26-21-mm2/arch/powerpc/kernel/traps.c
> ===
> --- lx26-21-mm2.orig/arch/powerpc/kernel/traps.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/powerpc/kernel/traps.c 2007-05-22 16:59:46.000000000 -0700
> @@ -190,7 +190,7 @@ void _exception(int signr, struct pt_reg
> * generate the same exception over and over again and we get
> * nowhere. Better to kill it and let the kernel panic.
> */
> - if (is_init(current)) {
> + if (is_global_init(current)) {
> __sighandler_t handler;
>
> spin_lock_irq(&current->sighand->siglock);
> Index: lx26-21-mm2/arch/powerpc/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/powerpc/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/powerpc/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -374,7 +374,7 @@ bad_area_nosemaphore:
> */
> out_of_memory:
> up_read(&mm->mmap_sem);
> - if (is_init(current)) {

> + if (is_global_init(current)) {
>   yield();
>   down_read(&mm->mmap_sem);
>   goto survive;
> Index: lx26-21-mm2/arch/powerpc/platforms/pseries/ras.c
> ===
> --- lx26-21-mm2.orig/arch/powerpc/platforms/pseries/ras.c 2007-05-22 16:58:38.000000000
-0700
> +++ lx26-21-mm2/arch/powerpc/platforms/pseries/ras.c 2007-05-22 16:59:46.000000000 -0700
> @@ -332,7 +332,7 @@ static int recover_mce(struct pt_regs *r
>   err->disposition == RTAS_DISP_NOT_RECOVERED &&
>   err->target == RTAS_TARGET_MEMORY &&
>   err->type == RTAS_TYPE_ECC_UNCORR &&
> -   !(current->pid == 0 || is_init(current))) {
> +   !(current->pid == 0 || is_global_init(current))) {
>   /* Kill off a user process with an ECC error */
>   printk(KERN_ERR "MCE: uncorrectable ecc error for pid %d\n",
>   current->pid);
> Index: lx26-21-mm2/arch/ppc/kernel/traps.c
> ===
> --- lx26-21-mm2.orig/arch/ppc/kernel/traps.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/ppc/kernel/traps.c 2007-05-22 16:59:46.000000000 -0700
> @@ -120,7 +120,7 @@ void _exception(int signr, struct pt_reg
>   * generate the same exception over and over again and we get
>   * nowhere. Better to kill it and let the kernel panic.
>   */
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>   __sighandler_t handler;
>
>   spin_lock_irq(&current->sighand->siglock);
> Index: lx26-21-mm2/arch/ppc/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/ppc/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/ppc/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -291,7 +291,7 @@ bad_area:
>   */
> out_of_memory:
>   up_read(&mm->mmap_sem);
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>   yield();
>   down_read(&mm->mmap_sem);
>   goto survive;
> Index: lx26-21-mm2/arch/s390/lib/uaccess_pt.c
> ===
> --- lx26-21-mm2.orig/arch/s390/lib/uaccess_pt.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/s390/lib/uaccess_pt.c 2007-05-22 16:59:46.000000000 -0700

> @@ -65,7 +65,7 @@ out:
>
> out_of_memory:
>   up_read(&mm->mmap_sem);
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>   yield();
>   down_read(&mm->mmap_sem);
>   goto survive;
> Index: lx26-21-mm2/arch/s390/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/s390/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/s390/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -211,7 +211,7 @@ static int do_out_of_memory(struct pt_re
>   struct mm_struct *mm = tsk->mm;
>
>   up_read(&mm->mmap_sem);
> - if (is_init(tsk)) {
> + if (is_global_init(tsk)) {
>   yield();
>   down_read(&mm->mmap_sem);
>   return 1;
> Index: lx26-21-mm2/arch/sh/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/sh/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/sh/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -233,7 +233,7 @@ no_context:
>   */
> out_of_memory:
>   up_read(&mm->mmap_sem);
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>   yield();
>   down_read(&mm->mmap_sem);
>   goto survive;
> Index: lx26-21-mm2/arch/sh64/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/sh64/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/sh64/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -276,7 +276,7 @@ bad_area:
>   show_regs(regs);
> #endif
>   }
> - if (is_init(tsk)) {
> + if (is_global_init(tsk)) {
>   panic("INIT had user mode bad_area\n");
>   }
>   tsk->thread.address = address;

```
> @@ -318,14 +318,14 @@ no_context:
>   * us unable to handle the page fault gracefully.
>   */
>  out_of_memory:
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>    panic("INIT out of memory\n");
>    yield();
>    goto survive;
>   }
>   printk("fault:Out of memory\n");
>   up_read(&mm->mmap_sem);
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>    yield();
>    down_read(&mm->mmap_sem);
>    goto survive;
> Index: lx26-21-mm2/arch/um/kernel/trap.c
> ===================================================================
> --- lx26-21-mm2.orig/arch/um/kernel/trap.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/um/kernel/trap.c 2007-05-22 16:59:46.000000000 -0700
> @@ -120,7 +120,7 @@ out_nosemaphore:
>   * us unable to handle the page fault gracefully.
>   */
>  out_of_memory:
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>    up_read(&mm->mmap_sem);
>    yield();
>    down_read(&mm->mmap_sem);
> Index: lx26-21-mm2/arch/x86_64/mm/fault.c
> ===================================================================
> --- lx26-21-mm2.orig/arch/x86_64/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/x86_64/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -223,7 +223,7 @@ static int is_errata93(struct pt_regs *r
>
>  int unhandled_signal(struct task_struct *tsk, int sig)
>  {
> - if (is_init(tsk))
> + if (is_global_init(tsk))
>    return 1;
>   if (tsk->ptrace & PT_PTRACED)
>    return 0;
> @@ -557,7 +557,7 @@ no_context:
>   */
>  out_of_memory:
>   up_read(&mm->mmap_sem);
> - if (is_init(current)) {
```

> + if (is_global_init(current)) {
>   yield();
>   goto again;
>   }
> Index: lx26-21-mm2/arch/xtensa/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/xtensa/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/xtensa/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -144,7 +144,7 @@ bad_area:
>   */
> out_of_memory:
>   up_read(&mm->mmap_sem);
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>   yield();
>   down_read(&mm->mmap_sem);
>   goto survive;
> Index: lx26-21-mm2/drivers/char/sysrq.c
> ===
> --- lx26-21-mm2.orig/drivers/char/sysrq.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/drivers/char/sysrq.c 2007-05-22 16:59:46.000000000 -0700
> @@ -250,7 +250,7 @@ static void send_sig_all(int sig)
>   struct task_struct *p;
>
>   for_each_process(p) {
> - if (p->mm && !is_init(p))
> + if (p->mm && !is_global_init(p))
>   /* Not swapper, init nor kernel thread */
>   force_sig(sig, p);
>   }
> Index: lx26-21-mm2/include/linux/sched.h
> ===
> --- lx26-21-mm2.orig/include/linux/sched.h 2007-05-22 16:59:44.000000000 -0700
> +++ lx26-21-mm2/include/linux/sched.h 2007-05-22 16:59:46.000000000 -0700
> @@ -1171,16 +1171,9 @@ static inline int pid_alive(struct task_
>   return p->pids[PIDTYPE_PID].pid != NULL;
> }
>
> -/**
> - * is_init - check if a task structure is init
> - * @tsk: Task structure to be checked.
> - *
> - * Check if a task structure is the first user space task the kernel created.
> - */
> -static inline int is_init(struct task_struct *tsk)
> -{
> - return tsk->pid == 1;
> -}

> +struct pid_namespace;
> +extern int is_global_init(struct task_struct *tsk);
> +extern int is_container_init(struct task_struct *tsk);
>
> extern struct pid *cad_pid;
>
> Index: lx26-21-mm2/kernel/capability.c
> ===
> --- lx26-21-mm2.orig/kernel/capability.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/kernel/capability.c 2007-05-22 16:59:46.000000000 -0700
> @@ -12,6 +12,7 @@
> #include <linux/module.h>
> #include <linux/security.h>
> #include <linux/syscalls.h>
> +#include <linux/pid_namespace.h>
> #include <asm/uaccess.h>
>
> unsigned securebits = SECUREBITS_DEFAULT; /* systemwide security settings */
> @@ -135,7 +136,7 @@ static inline int cap_set_all(kernel_cap
>   int found = 0;
>
>   do_each_thread(g, target) {
> -   if (target == current || is_init(target))
> +   if (target == current || is_container_init(target))

Ok, this is simply conforming to the capset(2) manpage/specs, so that is
correct. (as the only actual usage of is_container_init()).

>   continue;
>   found = 1;
>   if (security_capset_check(target, effective, inheritable,
> Index: lx26-21-mm2/kernel/exit.c
> ===
> --- lx26-21-mm2.orig/kernel/exit.c 2007-05-22 16:59:42.000000000 -0700
> +++ lx26-21-mm2/kernel/exit.c 2007-05-22 16:59:46.000000000 -0700
> @@ -230,7 +230,7 @@ static int will_become_orphaned_pgrp(str
>   do_each_pid_task(pgrp, PIDTYPE_PGID, p) {
>   if (p == ignored_task
> || p->exit_state
> - || is_init(p->real_parent))
> + || is_global_init(p->real_parent))
>   continue;
>   if (task_pgrp(p->real_parent) != pgrp &&
>   task_session(p->real_parent) == task_session(p)) {
> Index: lx26-21-mm2/kernel/kexec.c
> ===
> --- lx26-21-mm2.orig/kernel/kexec.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/kernel/kexec.c 2007-05-22 16:59:46.000000000 -0700

> @@ -42,7 +42,7 @@ struct resource crashk_res = {
>
> int kexec_should_crash(struct task_struct *p)
> {
> - if (in_interrupt() || !p->pid || is_init(p) || panic_on_oops)
> + if (in_interrupt() || !p->pid || is_global_init(p) || panic_on_oops)
>   return 1;
>   return 0;
> }
> Index: lx26-21-mm2/kernel/sysctl.c
> ===
> --- lx26-21-mm2.orig/kernel/sysctl.c 2007-05-22 16:59:41.000000000 -0700
> +++ lx26-21-mm2/kernel/sysctl.c 2007-05-22 16:59:46.000000000 -0700
> @@ -1730,7 +1730,7 @@ int proc_dointvec_bset(ctl_table *table,
>   return -EPERM;
>   }
>
> - op = is_init(current) ? OP_SET : OP_AND;
> + op = is_global_init(current) ? OP_SET : OP_AND;
>   return do_proc_dointvec(table,write,filp,buffer,lenp,ppos,
>   do_proc_dointvec_bset_conv,&op);
> }
> Index: lx26-21-mm2/mm/oom_kill.c
> ===
> --- lx26-21-mm2.orig/mm/oom_kill.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/mm/oom_kill.c 2007-05-22 16:59:46.000000000 -0700
> @@ -222,7 +222,7 @@ static struct task_struct *select_bad_pr
>   if (!p->mm)
>   continue;
>   /* skip the init task */
> - if (is_init(p))
> + if (is_global_init(p))
>   continue;
>
>   /*
> @@ -275,7 +275,7 @@ static struct task_struct *select_bad_pr
>   */
> static void __oom_kill_task(struct task_struct *p, int verbose)
> {
> - if (is_init(p)) {
> + if (is_global_init(p)) {
>   WARN_ON(1);
>   printk(KERN_WARNING "tried to kill init!\n");
>   return;
> Index: lx26-21-mm2/security/commoncap.c
> ===
> --- lx26-21-mm2.orig/security/commoncap.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/security/commoncap.c 2007-05-22 16:59:46.000000000 -0700

> @@ -306,7 +306,7 @@ void cap_bprm_apply_creds (struct linux_
>   /* For init, we want to retain the capabilities set
>   * in the init_task struct. Thus we skip the usual
>   * capability rules */
> - if (!is_init(current)) {
> + if (!is_global_init(current)) {

At some point we may want to make this more baroque to allow a container
init to keep all capabilities relative to it's own container. But we're
not there yet.

So actually this might be a good reason to finally repost the user
namespace patchset. Ironically I was going to do it during a lull in
containers@ traffic, and I'm doing it at perhaps the highest traffic
time ever, but it would be good to discuss the semantics of having
capabilities "to a namespace" and storing those to a namespace other
than your own in your keyring.

-serge

>   current->cap_permitted = new_permitted;
>   current->cap_effective =
>   cap_intersect (new_permitted, bprm->cap_effective);
> Index: lx26-21-mm2/arch/avr32/kernel/traps.c
> ===
> --- lx26-21-mm2.orig/arch/avr32/kernel/traps.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/avr32/kernel/traps.c 2007-05-22 16:59:46.000000000 -0700
> @@ -88,7 +88,7 @@ void _exception(long signr, struct pt_re
>   * generate the same exception over and over again and we get
>   * nowhere. Better to kill it and let the kernel panic.
>   */
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>   __sighandler_t handler;
>
>   spin_lock_irq(&current->sighand->siglock);
> Index: lx26-21-mm2/arch/avr32/mm/fault.c
> ===
> --- lx26-21-mm2.orig/arch/avr32/mm/fault.c 2007-05-22 16:58:38.000000000 -0700
> +++ lx26-21-mm2/arch/avr32/mm/fault.c 2007-05-22 16:59:46.000000000 -0700
> @@ -173,7 +173,7 @@ bad_area:
>   if (exception_trace)
>   printk("%s%s[%d]: segfault at %08lx pc %08lx "
>   "sp %08lx ecr %lu\n",
> -   is_init(tsk) ? KERN_EMERG : KERN_INFO,
> +   is_global_init(tsk) ? KERN_EMERG : KERN_INFO,
>   tsk->comm, tsk->pid, address, regs->pc,
>   regs->sp, ecr);

```
>    _exception(SIGSEGV, regs, code, address);
> @@ -222,7 +222,7 @@ no_context:
>     */
>  out_of_memory:
>    up_read(&mm->mmap_sem);
> - if (is_init(current)) {
> + if (is_global_init(current)) {
>     yield();
>     down_read(&mm->mmap_sem);
>     goto survive;
> @@ -244,7 +244,7 @@ do_sigbus:
>    if (exception_trace)
>     printk("%s%s[%d]: bus error at %08lx pc %08lx "
>            "sp %08lx ecr %lu\n",
> -          is_init(tsk) ? KERN_EMERG : KERN_INFO,
> +          is_global_init(tsk) ? KERN_EMERG : KERN_INFO,
>            tsk->comm, tsk->pid, address, regs->pc,
>            regs->sp, ecr);
>
> Index: lx26-21-mm2/kernel/pid.c
> ===================================================================
> --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:34.000000000 -0700
> +++ lx26-21-mm2/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
> @@ -71,6 +71,27 @@ struct pid_namespace init_pid_ns = {
>    .child_reaper = &init_task
>  };
>
> +
> +/**
> + * is_global_init - check if a task structure is init
> + * @tsk: Task structure to be checked.
> + *
> + * Check if a task structure is the first user space task the kernel created.
> + */
> +int is_global_init(struct task_struct *tsk)
> +{
> + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);
> +}
> +
> +/*
> + * is_container_init:
> + * check whether in the task is init in it's own pid namespace.
> + */
> +int is_container_init(struct task_struct *tsk)
> +{
> + return tsk->pid == 1;
> +}
> +
```

> /*
> * Note: disable interrupts while the pidmap_lock is held as an
> * interrupt might come in and do read_lock(&tasklist_lock).

This looks good. Thanks, Suka.

-serge

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

---

Subject: Re: [RFC][PATCH 15/16] Enable signaling child reaper from parent ns.
Posted by serue on Thu, 24 May 2007 15:59:40 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
>
> Subject: Enable signaling child reaper from parent ns.
>
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>
> The reaper of a child namespace must receive signals from its parent pid
> namespace but not receive any signals from its own namespace.
>
> This is a very early draft :-) and following tests seem to pass
>
> - Successfully kill child reaper from parent namespace (init_pid_ns)
>
> - Fail to kill child reaper from within its namespace (non init_pid_ns)
>
> - kill -1 1 from init_pid_ns seemed to work (rescanned inittab)
>
> TODO:
> - Test async io and SIGIO delivery.
>
> - Allow any legitimate signals that the child reaper can receive
>   from within its namespace? (we block all signals now)
>
>   - Sending SIGKILL to the child reaper of a namespace terminates the
>   namespace But if the namespace remounted /proc from user space,
>   /proc would remain mounted even after reaper and other process in
>   the namespace go away.
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> kernel/signal.c | 22 ++++++++++++++++++++-

> 1 file changed, 21 insertions(+), 1 deletion(-)
>
> Index: lx26-21-mm2/kernel/signal.c
> ===
> --- lx26-21-mm2.orig/kernel/signal.c 2007-05-22 16:59:42.000000000 -0700
> +++ lx26-21-mm2/kernel/signal.c 2007-05-22 16:59:57.000000000 -0700
> @@ -507,6 +507,20 @@ static int check_kill_permission(int sig
>   && !capable(CAP_KILL))
>   return error;
>
> + /*
> + * If t is the reaper of its namespace and someone from that
> + * namespace is trying to send a signal.
> + *
> + * Note: If some one from parent namespace is sending a signal,
> + *   task_child_reaper() != t and we allow the signal.
> + *
> + * In the child namespace, does this block even legitimate signals
> + * like the ones telinit sends to /sbin/init ?
> + *
> + */
> + if ((!is_global_init(t)) && (t == task_child_reaper(t)))

Couldn't you more clearly achieve what you want by doing:
 if ((!is_global_init(t)) && (t == task_child_reaper(current)))

Still like you say I think you need to study more how current code does
the right thing for the global init. Reproduce exactly that if t ==
task_child_reaper(current), else treat like any other task. And though
I said "reproduce", I should think you could do it without separate
checks as you have here.

-serge

> + return -EPERM;
> +
>   error = security_task_kill(t, info, sig, 0);
>   if (!error)
>   audit_signal_info(sig, t); /* Let audit system see the signal */
> @@ -1910,7 +1924,13 @@ relock:
>   /*
>   * Init of a pid space gets no signals it doesn't want from
>   * within that pid space. It can of course get signals from
> -   * its parent pid space.
> +   * its parent pid space. But we have no way of knowing the
> +   * namespace from which the signal was sent. For now check
> +   * if we are global init here and add additional checks in
> +   * sys_kill() and friends.

> + *
> + * Note that t == task_child_reaper(t) implies t is the global
> + * init (and we are in init_pid_ns).
>   */
>   if (current == task_child_reaper(current))
>   continue;

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by Dave Hansen on Thu, 24 May 2007 16:37:18 GMT
View Forum Message <> Reply to Message

On Thu, 2007-05-24 at 13:24 +0400, Pavel Emelianov wrote:
> > | > +int is_global_init(struct task_struct *tsk)
> > | > +{
> > | > + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);
> > |
> > | This can OOPS if you pass arbitrary task to this call...
> > | tsk->nsproxy can already be NULL.
> >
> > Hmm. You are right. btw, this could be a bisect issue. Patch 9 of uses
> > pid_ns from pid->upid_list and removes nsproxy->pid_ns.
>
> Yes, but that patch is not good either.
> task_pid(tsk) may become NULL as well and this will oops.

Have you reviewed the call paths to make sure this can actually happen
in practice?

This just seems like another one of those racing-with-task-exit races.
Shouldn't be too invasive to solve.

-- Dave

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

---

Subject: Re: [RFC][PATCH 14/16] Introduce proc_mnt for pid_ns
Posted by Dave Hansen on Thu, 24 May 2007 16:51:23 GMT
View Forum Message <> Reply to Message

On Thu, 2007-05-24 at 14:15 +0400, Pavel Emelianov wrote:
>
> > s->s_flags |= MS_NODIRATIME | MS_NOSUID | MS_NOEXEC;
> > @@ -466,6 +467,7 @@ int proc_fill_super(struct super_block *
> > s->s_magic = PROC_SUPER_MAGIC;
> > s->s_op = &proc_sops;
> > s->s_time_gran = 1;
> > + s->s_fs_info = pid_ns;
>
> One more thing I've just noticed - you don't get the namespace
> here so after all the tasks die and namespace is freed we
> have a proc mount pointing to freed namespace...

Yep, missed reference count. Thanks for catching this!

-- Dave

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by Pavel Emelianov on Fri, 25 May 2007 06:39:28 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2007-05-24 at 13:24 +0400, Pavel Emelianov wrote:
>>> | > +int is_global_init(struct task_struct *tsk)
>>> | > +{
>>> | > + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);
>>> |
>>> | This can OOPS if you pass arbitrary task to this call...
>>> | tsk->nsproxy can already be NULL.
>>>
>>> Hmm. You are right. btw, this could be a bisect issue. Patch 9 of uses
>>> pid_ns from pid->upid_list and removes nsproxy->pid_ns.
>> Yes, but that patch is not good either.
>> task_pid(tsk) may become NULL as well and this will oops.
>
> Have you reviewed the call paths to make sure this can actually happen
> in practice?
>
> This just seems like another one of those racing-with-task-exit races.
> Shouldn't be too invasive to solve.

It is, but if we make patch that OOPSes the kernel in 0.1%

of cases and we do know this - this MUST be fixed.

> -- Dave
>
>

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

---

Subject: Re: [RFC][PATCH 07/16] Move alloc_pid call to copy_process
Posted by Pavel Emelianov on Fri, 25 May 2007 06:41:58 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Subject: Move alloc_pid call to copy_process
>
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>
> Move alloc_pid() into copy_process(). This will keep all pid and pid
> namespace code together and simplify error handling when we support
> multiple pid namespaces.

I haven't found this in patches, so I ask it here:

We clone a new task with CLONE_NEWPIDS flag. This task
allocates its PIDTYPE_PID pid and this pid happens in
booth parent and child namespace. This is OK. Then new
task attaches PIDTYPE_SID and PIDTYPE_PGID pids from parent
task. But these ones are in parent namespace only.

Right? Is that good?

> Changelog:
> - [Eric Biederman] Move the check of copy_process_type to alloc_pid()/
> free_pid() and to avoid clutter in copy_process().
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> include/linux/pid.h | 7 ++++++-
> kernel/fork.c | 21 +++++++++++++++--------
> kernel/pid.c | 10 +++++++++-
> 3 files changed, 28 insertions(+), 10 deletions(-)
>
> Index: lx26-21-mm2/include/linux/pid.h
> ==========

> --- lx26-21-mm2.orig/include/linux/pid.h 2007-05-22 16:59:40.000000000 -0700
> +++ lx26-21-mm2/include/linux/pid.h 2007-05-22 17:06:48.000000000 -0700
> @@ -3,6 +3,11 @@
>
> #include <linux/rcupdate.h>
>
> +enum copy_process_type {
> + COPY_NON_IDLE_PROCESS,
> + COPY_IDLE_PROCESS,
> +};
> +
> enum pid_type
> {
>   PIDTYPE_PID,
> @@ -95,7 +100,7 @@ extern struct pid *FASTCALL(find_pid(int
> 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(enum copy_process_type);
> extern void FASTCALL(free_pid(struct pid *pid));
>
> static inline pid_t pid_to_nr(struct pid *pid)
> Index: lx26-21-mm2/kernel/fork.c
> ===
> --- lx26-21-mm2.orig/kernel/fork.c 2007-05-22 16:59:41.000000000 -0700
> +++ lx26-21-mm2/kernel/fork.c 2007-05-22 17:06:48.000000000 -0700
> @@ -961,10 +961,11 @@ static struct task_struct *copy_process(
>   unsigned long stack_size,
>   int __user *parent_tidptr,
>   int __user *child_tidptr,
> -   struct pid *pid)
> +   enum copy_process_type copy_src)
> {
>   int retval;
>   struct task_struct *p = NULL;
> + struct pid *pid;
>
>   if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
>   return ERR_PTR(-EINVAL);
> @@ -1025,6 +1026,10 @@ static struct task_struct *copy_process(
>   if (p->binfmt && !try_module_get(p->binfmt->module))
>   goto bad_fork_cleanup_put_domain;
>
> + pid = alloc_pid(copy_src);
> + 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);
> @@ -1305,6 +1310,8 @@ bad_fork_cleanup_cpuset:
>  #endif
>   cpuset_exit(p);
>   delayacct_tsk_free(p);
> + free_pid(pid);
> +bad_fork_put_binfmt_module:
>   if (p->binfmt)
>    module_put(p->binfmt->module);
>  bad_fork_cleanup_put_domain:
> @@ -1331,7 +1338,7 @@ struct task_struct * __cpuinit fork_idle
>   struct pt_regs regs;
>
>   task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL, NULL,
> -    &init_struct_pid);
> +    COPY_IDLE_PROCESS);
>   if (!IS_ERR(task))
>    init_idle(task, cpu);
>
> @@ -1369,19 +1376,16 @@ long do_fork(unsigned long clone_flags,
>  {
>   struct task_struct *p;
>   int trace = 0;
> - struct pid *pid = alloc_pid();
>   long nr;
>
> - if (!pid)
> -  return -EAGAIN;
> - nr = pid->nr;
>   if (unlikely(current->ptrace)) {
>    trace = fork_traceflag (clone_flags);
>    if (trace)
>     clone_flags |= CLONE_PTRACE;
>   }
>
> - p = copy_process(clone_flags, stack_start, regs, stack_size, parent_tidptr, child_tidptr, pid);
> + p = copy_process(clone_flags, stack_start, regs, stack_size,
> +    parent_tidptr, child_tidptr, COPY_NON_IDLE_PROCESS);
>   /*
>    * Do this prior waking up the new thread - the thread pointer
>    * might get invalid after that point, if the thread exits quickly.
> @@ -1389,6 +1393,8 @@ long do_fork(unsigned long clone_flags,
>   if (!IS_ERR(p)) {
>    struct completion vfork;
>
> +  nr = pid_to_nr(task_pid(p));
```

```
> +
>    if (clone_flags & CLONE_VFORK) {
>     p->vfork_done = &vfork;
>     init_completion(&vfork);
> @@ -1422,7 +1428,6 @@ long do_fork(unsigned long clone_flags,
>     }
>    }
>   } else {
> -  free_pid(pid);
>    nr = PTR_ERR(p);
>   }
>   return nr;
> Index: lx26-21-mm2/kernel/pid.c
> ===================================================================
> --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
> +++ lx26-21-mm2/kernel/pid.c 2007-05-22 17:06:48.000000000 -0700
> @@ -216,6 +216,10 @@ fastcall void free_pid(struct pid *pid)
>   /* We can be called with write_lock_irq(&tasklist_lock) held */
>   unsigned long flags;
>
> + /* check this here to keep copy_process() cleaner */
> + if (unlikely(pid == &init_struct_pid))
> +  return;
> +
>   spin_lock_irqsave(&pidmap_lock, flags);
>   hlist_del_rcu(&pid->pid_chain);
>   spin_unlock_irqrestore(&pidmap_lock, flags);
> @@ -224,12 +228,16 @@ fastcall void free_pid(struct pid *pid)
>   call_rcu(&pid->rcu, delayed_put_pid);
>  }
>
> -struct pid *alloc_pid(void)
> +struct pid *alloc_pid(enum copy_process_type copy_src)
>  {
>   struct pid *pid;
>   enum pid_type type;
>   int nr = -1;
>
> + /* check this here to keep copy_process() cleaner */
> + if (unlikely(copy_src == COPY_IDLE_PROCESS))
> +  return &init_struct_pid;
> +
>   pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
>   if (!pid)
>    goto out;
> _____________________________________________
> Containers mailing list
> Containers@lists.linux-foundation.org
```

> https://lists.linux-foundation.org/mailman/listinfo/containers
>

_____

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

---

Subject: Re: [RFC][PATCH 14/16] Introduce proc_mnt for pid_ns
Posted by Pavel Emelianov on Fri, 25 May 2007 12:27:54 GMT
View Forum Message <> Reply to Message

Dave Hansen wrote:
> On Thu, 2007-05-24 at 14:15 +0400, Pavel Emelianov wrote:
>>> s->s_flags |= MS_NODIRATIME | MS_NOSUID | MS_NOEXEC;
>>> @@ -466,6 +467,7 @@ int proc_fill_super(struct super_block *
>>> s->s_magic = PROC_SUPER_MAGIC;
>>> s->s_op = &proc_sops;
>>> s->s_time_gran = 1;
>>> + s->s_fs_info = pid_ns;
>> One more thing I've just noticed - you don't get the namespace
>> here so after all the tasks die and namespace is freed we
>> have a proc mount pointing to freed namespace...
>
> Yep, missed reference count. Thanks for catching this!

refcount is not the only badness. As I said you have
race in testing superblock's namespace and setting it
and one more: do you use one struct proc_dir_entry
proc_root for all the superolocks?

> -- Dave
>
>

_____

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by Dave Hansen on Fri, 25 May 2007 16:18:49 GMT
View Forum Message <> Reply to Message

On Fri, 2007-05-25 at 10:39 +0400, Pavel Emelianov wrote:
>

> It is, but if we make patch that OOPSes the kernel in 0.1%
> of cases and we do know this - this MUST be fixed.

Oh, yeah, it certainly needs to be fixed. I just wanted to make sure
you weren't seeing some endemic problem with the patches.

-- Dave

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

---

Subject: Re: [RFC][PATCH 14/16] Introduce proc_mnt for pid_ns
Posted by Dave Hansen on Fri, 25 May 2007 16:21:16 GMT
View Forum Message <> Reply to Message

On Fri, 2007-05-25 at 16:27 +0400, Pavel Emelianov wrote:
>
> refcount is not the only badness. As I said you have
> race in testing superblock's namespace and setting it
> and one more: do you use one struct proc_dir_entry
> proc_root for all the superolocks?

Yep. Do you see some problems with that?

-- Dave

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

---

Subject: Re: [RFC][PATCH 15/16] Enable signaling child reaper from parent ns.
Posted by serue on Fri, 25 May 2007 20:13:33 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
>
> Subject: Enable signaling child reaper from parent ns.
>
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>
> The reaper of a child namespace must receive signals from its parent pid
> namespace but not receive any signals from its own namespace.

>
> This is a very early draft :-) and following tests seem to pass
>
> - Successfully kill child reaper from parent namespace (init_pid_ns)
>
> - Fail to kill child reaper from within its namespace (non init_pid_ns)
>
> - kill -1 1 from init_pid_ns seemed to work (rescanned inittab)
>
> TODO:
> - Test async io and SIGIO delivery.
>
> - Allow any legitimate signals that the child reaper can receive
>   from within its namespace? (we block all signals now)
>
>   - Sending SIGKILL to the child reaper of a namespace terminates the
>   namespace But if the namespace remounted /proc from user space,
>   /proc would remain mounted even after reaper and other process in
>   the namespace go away.
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> kernel/signal.c | 22 ++++++++++++++++++++-
> 1 file changed, 21 insertions(+), 1 deletion(-)
>
> Index: lx26-21-mm2/kernel/signal.c
> ==========
> --- lx26-21-mm2.orig/kernel/signal.c 2007-05-22 16:59:42.000000000 -0700
> +++ lx26-21-mm2/kernel/signal.c 2007-05-22 16:59:57.000000000 -0700
> @@ -507,6 +507,20 @@ static int check_kill_permission(int sig
>   && !capable(CAP_KILL))
>   return error;
>
> + /*
> + * If t is the reaper of its namespace and someone from that
> + * namespace is trying to send a signal.
> + *
> + * Note: If some one from parent namespace is sending a signal,
> + *   task_child_reaper() != t and we allow the signal.
> + *
> + * In the child namespace, does this block even legitimate signals
> + * like the ones telinit sends to /sbin/init ?
> + *
> + */
> + if ((!is_global_init(t)) && (t == task_child_reaper(t)))
> + return -EPERM;

Ok, let's just go over the desired semantics.

Current treatment of init for signals is evident at
kernel/signal.c:get_signal_to_deliver(). There we used to check whether
current->pid == 1, which was turned into (current == child_reaper(current)).

First note on that, (current == child_reaper(current)) is correct if we
want to treat a container init as init no matter who sends the signal,
however I contend that if a signal is sent from an ancester namespace,
then we want to treat the task like any other task.

Could I get some confirmation or rebuttal from Suka, Eric, Dave, or
Pavel?

Next, note where that check is done: If the task has it's own custom
handler for a signal, then that will get called. Only if the handler is
the default, do we check whether the target is the init task. This is
why this patch is wrong, Suka. If init sets up a handler for USR1, then
tasks should be able to do a kill -USR1 1.

Ok, so finally here's the problem to solve. As I said above, if we are
willing to treat a container init like init no matter who signals it,
then we're ok with the current code. But if we want a container init to
be treated like a normal process if signaled from an ancestor pidns,
then we'll need to tack a struct pid or struct pid_ns pointer into
struct siginfo. And this means taking a reference to one of those,
which means slowing things down a touch.

thanks,
-serge

```
> +
>   error = security_task_kill(t, info, sig, 0);
>   if (!error)
>    audit_signal_info(sig, t); /* Let audit system see the signal */
> @@ -1910,7 +1924,13 @@ relock:
>    /*
>     * Init of a pid space gets no signals it doesn't want from
>     * within that pid space. It can of course get signals from
> -   * its parent pid space.
> +   * its parent pid space. But we have no way of knowing the
> +   * namespace from which the signal was sent. For now check
> +   * if we are global init here and add additional checks in
> +   * sys_kill() and friends.
> +   *
> +   * Note that t == task_child_reaper(t) implies t is the global
> +   * init (and we are in init_pid_ns).
>     */
```

> if (current == task_child_reaper(current))
> continue;

_____

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by Sukadev Bhattiprolu on Fri, 25 May 2007 20:44:45 GMT
View Forum Message <> Reply to Message

Dave Hansen [hansendc@us.ibm.com] wrote:
| On Thu, 2007-05-24 at 13:24 +0400, Pavel Emelianov wrote:
| > > | > +int is_global_init(struct task_struct *tsk)
| > > | > +{
| > > | > + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);
| > > |
| > > | This can OOPS if you pass arbitrary task to this call...
| > > | tsk->nsproxy can already be NULL.
| > >
| > > Hmm. You are right. btw, this could be a bisect issue. Patch 9 of uses
| > > pid_ns from pid->upid_list and removes nsproxy->pid_ns.
| >
| > Yes, but that patch is not good either.
| > task_pid(tsk) may become NULL as well and this will oops.
|
| Have you reviewed the call paths to make sure this can actually happen
| in practice?

task_pid() can be NULL when we are tearing down the task structure in
release_task() and in the tiny window between detach_pid() and attach_pid()
in de_thread().

I think task_pid() is safe as long as it is called for 'current'. (we should
probably add some comments)

I will double check my code, but I think all my calls to task_pid() and hence,
to task_active_pid_ns() are safe, except for two cases:

a) is_global_init(). There are a few calls to process other than
current, but not sure if they are a problem.

For instance in current code, unhandled_signal() checks
tsk->pid == 1 and proceeds to derefernce tsk->sighand.

If task_pid() is NULL because the task was in release_task(),
then so is tsk->sighand.

b) the temporary check I added in check_kill_permissions().
(I need to address Serge's comment here anyway).

To make is_global_init() more efficient and independent of task_pid(),
can we steal a bit from task_struct->flags ? Like PF_KSWAPD, and there
are unused bits :-)

|
| This just seems like another one of those racing-with-task-exit races.
| Shouldn't be too invasive to solve.

A little invasive approach for the release_task() case could be to remove
the 'struct pid' from the hash table, but leave it attached to the
'task_struct' till the 'task_struct' itself is freed.

Removing from hash table ensures no one finds this process anymore, but
keeping it attached allows those who have already found the 'task_struct'
to also use the 'struct pid' as long as they have the task_struct.

Of course, needs investigation and micro surgery.

|
| -- Dave

Subject: Re: [RFC][PATCH 04/16] Use pid_to_nr() in process info functions
Posted by Sukadev Bhattiprolu on Fri, 25 May 2007 22:53:18 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@sw.ru] wrote:
| > ===
| > --- lx26-21-mm2.orig/kernel/timer.c 2007-05-22 16:58:38.000000000 -0700
| > +++ lx26-21-mm2/kernel/timer.c 2007-05-22 16:59:44.000000000 -0700
| > @@ -945,7 +945,7 @@ asmlinkage long sys_getppid(void)
| > int pid;
| >
| > rcu_read_lock();
| > - pid = rcu_dereference(current->real_parent)->tgid;
| > + pid = pid_to_nr(task_parent_tgid(current));
|
| This breaks fsys_getppid() call in ia64...

Good catch. I need to figure out how to make the pid_to_nr()

call in IA64 assembly and if there are other architectures
that implement in assembly.

But in general, do you see any problems with the interfaces
like task_parent_tgid() ?

|
| > rcu_read_unlock();
| >
| > return pid;

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

---

Subject: Re: [RFC][PATCH 07/16] Move alloc_pid call to copy_process
Posted by Sukadev Bhattiprolu on Fri, 25 May 2007 23:01:59 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| sukadev@us.ibm.com wrote:
| > Subject: Move alloc_pid call to copy_process
| >
| > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
| >
| > Move alloc_pid() into copy_process(). This will keep all pid and pid
| > namespace code together and simplify error handling when we support
| > multiple pid namespaces.
|
| I haven't found this in patches, so I ask it here:
|
| We clone a new task with CLONE_NEWPIDS flag. This task
| allocates its PIDTYPE_PID pid and this pid happens in
| booth parent and child namespace. This is OK.

| Then new task attaches PIDTYPE_SID and PIDTYPE_PGID pids from
| parent task. But these ones are in parent namespace only.
|
| Right? Is that good?
|

In this patch, yes, we still attach to the parent process, bc at this
point we still support only one namespace.

In the patch that actually allows creating multiple namespaces,
(Patch #11), I have the following code which makes the process
that cloned its pid ns a session and pgrp leader, just like

/sbin/init for init_pid_ns.

```
@@ -1255,11 +1254,17 @@ static struct task_struct *copy_process(
                        __ptrace_link(p, current->parent);

                if (thread_group_leader(p)) {
+                       struct pid *pgrp = task_pgrp(current);
+                       struct pid *session = task_session(current);
+
+                       if (clone_flags & CLONE_NEWPID)
+                               pgrp = session = pid;
+
                        p->signal->tty = current->signal->tty;
-                       p->signal->pgrp = process_group(current);
-                       set_signal_session(p->signal, process_session(current));
-                       attach_pid(p, PIDTYPE_PGID, task_pgrp(current));
-                       attach_pid(p, PIDTYPE_SID, task_session(current));
+                       p->signal->pgrp = pid_to_nr(pgrp);
+                       set_signal_session(p->signal, pid_to_nr(session));
+                       attach_pid(p, PIDTYPE_PGID, pgrp);
+                       attach_pid(p, PIDTYPE_SID, session);

                        list_add_tail_rcu(&p->tasks, &init_task.tasks);
                        __get_cpu_var(process_counts)++;

| > Changelog:
| >  - [Eric Biederman] Move the check of copy_process_type to alloc_pid()/
| >    free_pid() and to avoid clutter in copy_process().
| >
| > Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
| > ---
| >  include/linux/pid.h |    7 ++++++-
| >  kernel/fork.c       |   21 +++++++++++++++--------
| >  kernel/pid.c        |   10 +++++++++-
| >  3 files changed, 28 insertions(+), 10 deletions(-)
| >
| > Index: lx26-21-mm2/include/linux/pid.h
| > ===================================================================
| > --- lx26-21-mm2.orig/include/linux/pid.h 2007-05-22 16:59:40.000000000 -0700
| > +++ lx26-21-mm2/include/linux/pid.h 2007-05-22 17:06:48.000000000 -0700
| > @@ -3,6 +3,11 @@
| >
| >  #include <linux/rcupdate.h>
| >
| > +enum copy_process_type {
| > + COPY_NON_IDLE_PROCESS,
| > + COPY_IDLE_PROCESS,
| > +};
```

```
| > +
| >  enum pid_type
| >  {
| >   PIDTYPE_PID,
| > @@ -95,7 +100,7 @@ extern struct pid *FASTCALL(find_pid(int
| >  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(enum copy_process_type);
| >  extern void FASTCALL(free_pid(struct pid *pid));
| >
| >  static inline pid_t pid_to_nr(struct pid *pid)
| > Index: lx26-21-mm2/kernel/fork.c
| > ===================================================================
| > --- lx26-21-mm2.orig/kernel/fork.c 2007-05-22 16:59:41.000000000 -0700
| > +++ lx26-21-mm2/kernel/fork.c 2007-05-22 17:06:48.000000000 -0700
| > @@ -961,10 +961,11 @@ static struct task_struct *copy_process(
| >       unsigned long stack_size,
| >       int __user *parent_tidptr,
| >       int __user *child_tidptr,
| > -     struct pid *pid)
| > +     enum copy_process_type copy_src)
| >  {
| >   int retval;
| >   struct task_struct *p = NULL;
| > + struct pid *pid;
| >
| >   if ((clone_flags & (CLONE_NEWNS|CLONE_FS)) == (CLONE_NEWNS|CLONE_FS))
| >    return ERR_PTR(-EINVAL);
| > @@ -1025,6 +1026,10 @@ static struct task_struct *copy_process(
| >   if (p->binfmt && !try_module_get(p->binfmt->module))
| >    goto bad_fork_cleanup_put_domain;
| >
| > + pid = alloc_pid(copy_src);
| > + 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);
| > @@ -1305,6 +1310,8 @@ bad_fork_cleanup_cpuset:
| >  #endif
| >   cpuset_exit(p);
| >   delayacct_tsk_free(p);
| > + free_pid(pid);
| > +bad_fork_put_binfmt_module:
| >   if (p->binfmt)
```

```
| >    module_put(p->binfmt->module);
| >  bad_fork_cleanup_put_domain:
| > @@ -1331,7 +1338,7 @@ struct task_struct * __cpuinit fork_idle
| >   struct pt_regs regs;
| >
| >   task = copy_process(CLONE_VM, 0, idle_regs(&regs), 0, NULL, NULL,
| > -     &init_struct_pid);
| > +     COPY_IDLE_PROCESS);
| >   if (!IS_ERR(task))
| >    init_idle(task, cpu);
| >
| > @@ -1369,19 +1376,16 @@ long do_fork(unsigned long clone_flags,
| >  {
| >   struct task_struct *p;
| >   int trace = 0;
| > - struct pid *pid = alloc_pid();
| >   long nr;
| >
| > - if (!pid)
| > -  return -EAGAIN;
| > - nr = pid->nr;
| >   if (unlikely(current->ptrace)) {
| >    trace = fork_traceflag (clone_flags);
| >    if (trace)
| >     clone_flags |= CLONE_PTRACE;
| >   }
| >
| > - p = copy_process(clone_flags, stack_start, regs, stack_size, parent_tidptr, child_tidptr, pid);
| > + p = copy_process(clone_flags, stack_start, regs, stack_size,
| > +   parent_tidptr, child_tidptr, COPY_NON_IDLE_PROCESS);
| >   /*
| >    * Do this prior waking up the new thread - the thread pointer
| >    * might get invalid after that point, if the thread exits quickly.
| > @@ -1389,6 +1393,8 @@ long do_fork(unsigned long clone_flags,
| >   if (!IS_ERR(p)) {
| >    struct completion vfork;
| >
| > +  nr = pid_to_nr(task_pid(p));
| > +
| >    if (clone_flags & CLONE_VFORK) {
| >     p->vfork_done = &vfork;
| >     init_completion(&vfork);
| > @@ -1422,7 +1428,6 @@ long do_fork(unsigned long clone_flags,
| >     }
| >    }
| >   } else {
| > -  free_pid(pid);
| >    nr = PTR_ERR(p);
```

```
| >   }
| >   return nr;
| > Index: lx26-21-mm2/kernel/pid.c
| > ===================================================================
| > --- lx26-21-mm2.orig/kernel/pid.c 2007-05-22 16:59:46.000000000 -0700
| > +++ lx26-21-mm2/kernel/pid.c 2007-05-22 17:06:48.000000000 -0700
| > @@ -216,6 +216,10 @@ fastcall void free_pid(struct pid *pid)
| >   /* We can be called with write_lock_irq(&tasklist_lock) held */
| >   unsigned long flags;
| >
| > + /* check this here to keep copy_process() cleaner */
| > + if (unlikely(pid == &init_struct_pid))
| > +  return;
| > +
| >   spin_lock_irqsave(&pidmap_lock, flags);
| >   hlist_del_rcu(&pid->pid_chain);
| >   spin_unlock_irqrestore(&pidmap_lock, flags);
| > @@ -224,12 +228,16 @@ fastcall void free_pid(struct pid *pid)
| >   call_rcu(&pid->rcu, delayed_put_pid);
| >  }
| >
| > -struct pid *alloc_pid(void)
| > +struct pid *alloc_pid(enum copy_process_type copy_src)
| >  {
| >   struct pid *pid;
| >   enum pid_type type;
| >   int nr = -1;
| >
| > + /* check this here to keep copy_process() cleaner */
| > + if (unlikely(copy_src == COPY_IDLE_PROCESS))
| > +  return &init_struct_pid;
| > +
| >   pid = kmem_cache_alloc(pid_cachep, GFP_KERNEL);
| >   if (!pid)
| >    goto out;
| > _____________________________________________
| > Containers mailing list
| > Containers@lists.linux-foundation.org
| > https://lists.linux-foundation.org/mailman/listinfo/containers
| >
| > _____________________________________________
| > Devel mailing list
| > Devel@openvz.org
| > https://openvz.org/mailman/listinfo/devel
| >

_____________________________________________
Containers mailing list
Containers@lists.linux-foundation.org
```

Subject: Re: [RFC][PATCH 15/16] Enable signaling child reaper from parent ns.
Posted by Sukadev Bhattiprolu on Fri, 25 May 2007 23:11:46 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn [serue@us.ibm.com] wrote:
| Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
| >
| > Subject: Enable signaling child reaper from parent ns.
| >
| > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
| >
| > The reaper of a child namespace must receive signals from its parent pid
| > namespace but not receive any signals from its own namespace.
| >
| > This is a very early draft :-) and following tests seem to pass
| >
| > - Successfully kill child reaper from parent namespace (init_pid_ns)
| >
| > - Fail to kill child reaper from within its namespace (non init_pid_ns)
| >
| > - kill -1 1 from init_pid_ns seemed to work (rescanned inittab)
| >
| > TODO:
| > - Test async io and SIGIO delivery.
| >
| > - Allow any legitimate signals that the child reaper can receive
| >   from within its namespace? (we block all signals now)
| >
| >   - Sending SIGKILL to the child reaper of a namespace terminates the
| >   namespace But if the namespace remounted /proc from user space,
| >   /proc would remain mounted even after reaper and other process in
| >   the namespace go away.
| >
| > Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
| > ---
| > kernel/signal.c | 22 ++++++++++++++++++++-
| > 1 file changed, 21 insertions(+), 1 deletion(-)
| >
| > Index: lx26-21-mm2/kernel/signal.c
| > ===
| > --- lx26-21-mm2.orig/kernel/signal.c 2007-05-22 16:59:42.000000000 -0700
| > +++ lx26-21-mm2/kernel/signal.c 2007-05-22 16:59:57.000000000 -0700
| > @@ -507,6 +507,20 @@ static int check_kill_permission(int sig
| >   && !capable(CAP_KILL))
| >   return error;

| >
| > + /*
| > + * If t is the reaper of its namespace and someone from that
| > + * namespace is trying to send a signal.
| > + *
| > + * Note: If some one from parent namespace is sending a signal,
| > + *   task_child_reaper() != t and we allow the signal.
| > + *
| > + * In the child namespace, does this block even legitimate signals
| > + * like the ones telinit sends to /sbin/init ?
| > + *
| > + */
| > + if ((!is_global_init(t)) && (t == task_child_reaper(t)))
|
| Couldn't you more clearly achieve what you want by doing:
| if ((!is_global_init(t)) && (t == task_child_reaper(current)))

Yes. I think so. My current implementation of task_child_reaper() returns
reaper of current namespace, if called from within the namespace.

I still need to modify it like you mentioned in the other mail. I will
then go over this signals checks again. It was just a quick fix to allow
terminating the ns from ancestor ns.

|
| Still like you say I think you need to study more how current code does
| the right thing for the global init. Reproduce exactly that if t ==
| task_child_reaper(current), else treat like any other task. And though
| I said "reproduce", I should think you could do it without separate
| checks as you have here.
|
| -serge
|
| > + return -EPERM;
| > +
| >   error = security_task_kill(t, info, sig, 0);
| >   if (!error)
| >   audit_signal_info(sig, t); /* Let audit system see the signal */
| > @@ -1910,7 +1924,13 @@ relock:
| >   /*
| >   * Init of a pid space gets no signals it doesn't want from
| >   * within that pid space. It can of course get signals from
| > -   * its parent pid space.
| > +   * its parent pid space. But we have no way of knowing the
| > +   * namespace from which the signal was sent. For now check
| > +   * if we are global init here and add additional checks in
| > +   * sys_kill() and friends.
| > +   *

| > +   * Note that t == task_child_reaper(t) implies t is the global
| > +   * init (and we are in init_pid_ns).
| >   */
| >   if (current == task_child_reaper(current))
| >   continue;

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

---

Subject: Re: [RFC][PATCH 15/16] Enable signaling child reaper from parent ns.
Posted by Sukadev Bhattiprolu on Fri, 25 May 2007 23:18:11 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn [serue@us.ibm.com] wrote:
| Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
| >
| > Subject: Enable signaling child reaper from parent ns.
| >
| > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
| >
| > The reaper of a child namespace must receive signals from its parent pid
| > namespace but not receive any signals from its own namespace.
| >
| > This is a very early draft :-) and following tests seem to pass
| >
| > - Successfully kill child reaper from parent namespace (init_pid_ns)
| >
| > - Fail to kill child reaper from within its namespace (non init_pid_ns)
| >
| > - kill -1 1 from init_pid_ns seemed to work (rescanned inittab)
| >
| > TODO:
| > - Test async io and SIGIO delivery.
| >
| > - Allow any legitimate signals that the child reaper can receive
| >   from within its namespace? (we block all signals now)
| >
| >   - Sending SIGKILL to the child reaper of a namespace terminates the
| >   namespace But if the namespace remounted /proc from user space,
| >   /proc would remain mounted even after reaper and other process in
| >   the namespace go away.
| >
| > Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
| > ---
| > kernel/signal.c | 22 ++++++++++++++++++++-
| > 1 file changed, 21 insertions(+), 1 deletion(-)

| >
| > Index: lx26-21-mm2/kernel/signal.c
| > ===
| > --- lx26-21-mm2.orig/kernel/signal.c 2007-05-22 16:59:42.000000000 -0700
| > +++ lx26-21-mm2/kernel/signal.c 2007-05-22 16:59:57.000000000 -0700
| > @@ -507,6 +507,20 @@ static int check_kill_permission(int sig
| >   && !capable(CAP_KILL))
| >   return error;
| >
| > + /*
| > + * If t is the reaper of its namespace and someone from that
| > + * namespace is trying to send a signal.
| > + *
| > + * Note: If some one from parent namespace is sending a signal,
| > + *   task_child_reaper() != t and we allow the signal.
| > + *
| > + * In the child namespace, does this block even legitimate signals
| > + * like the ones telinit sends to /sbin/init ?
| > + *
| > + */
| > + if ((!is_global_init(t)) && (t == task_child_reaper(t)))
| > + return -EPERM;
|
| Ok, let's just go over the desired semantics.
|
| Current treatment of init for signals is evident at
| kernel/signal.c:get_signal_to_deliver(). There we used to check whether
| current->pid == 1, which was turned into (current == child_reaper(current)).
|
| First note on that, (current == child_reaper(current)) is correct if we
| want to treat a container init as init no matter who sends the signal,
| however I contend that if a signal is sent from an ancester namespace,
| then we want to treat the task like any other task.

I agree that we need to be able terminate the namespace from an ancestor
namespace. The container init should be special only to other process in
that namespace, its like any other process in the ancestor namespaces.

|
| Could I get some confirmation or rebuttal from Suka, Eric, Dave, or
| Pavel?
|
| Next, note where that check is done: If the task has it's own custom
| handler for a signal, then that will get called. Only if the handler is
| the default, do we check whether the target is the init task. This is
| why this patch is wrong, Suka. If init sets up a handler for USR1, then
| tasks should be able to do a kill -USR1 1.
|

| Ok, so finally here's the problem to solve. As I said above, if we are
| willing to treat a container init like init no matter who signals it,
| then we're ok with the current code. But if we want a container init to
| be treated like a normal process if signaled from an ancestor pidns,
| then we'll need to tack a struct pid or struct pid_ns pointer into
| struct siginfo. And this means taking a reference to one of those,
| which means slowing things down a touch.

I think we need the reference to struct pid in 'struct siginfo', but
again need to investigate more.

|
| thanks,
| -serge
|
|
| > +
| >   error = security_task_kill(t, info, sig, 0);
| >   if (!error)
| >   audit_signal_info(sig, t); /* Let audit system see the signal */
| > @@ -1910,7 +1924,13 @@ relock:
| >   /*
| >   * Init of a pid space gets no signals it doesn't want from
| >   * within that pid space. It can of course get signals from
| > -   * its parent pid space.
| > +   * its parent pid space. But we have no way of knowing the
| > +   * namespace from which the signal was sent. For now check
| > +   * if we are global init here and add additional checks in
| > +   * sys_kill() and friends.
| > +   *
| > +   * Note that t == task_child_reaper(t) implies t is the global
| > +   * init (and we are in init_pid_ns).
| >   */
| >   if (current == task_child_reaper(current))
| >   continue;

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

---

Subject: Re: [RFC][PATCH 0/16] Enable cloning of pid namespace
Posted by Pavel Emelianov on Tue, 29 May 2007 10:29:06 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Dave, Serge,
>

> Here is my current pid namespace patchset. There are still a couple of
> issues I am investigating, but appreciate any feedback.
>
> Suka
>

I tried to compile your patches with CONFIG_PID_NS=n and got this:

  CC   kernel/configs.o
kernel/pid.c:361: warning: function declaration isn't a prototype
kernel/pid.c: In function `dup_struct_pid':
kernel/pid.c:489: warning: initialization makes pointer from integer without a cast

Thanks,
Pavel.

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

---

Subject: Re: [RFC][PATCH 0/16] Enable cloning of pid namespace
Posted by Dave Hansen on Tue, 29 May 2007 15:48:09 GMT
View Forum Message <> Reply to Message

On Tue, 2007-05-29 at 14:29 +0400, Pavel Emelianov wrote:
> sukadev@us.ibm.com wrote:
> > Dave, Serge,
> >
> > Here is my current pid namespace patchset. There are still a couple of
> > issues I am investigating, but appreciate any feedback.
> >
> > Suka
> >
>
> I tried to compile your patches with CONFIG_PID_NS=n and got this:
>
>   CC   kernel/configs.o
> kernel/pid.c:361: warning: function declaration isn't a prototype
> kernel/pid.c: In function `dup_struct_pid':
> kernel/pid.c:489: warning: initialization makes pointer from integer without a cast

Could you send along your actual .config?

-- Dave

_____
Containers mailing list

Subject: Re: [RFC][PATCH 15/16] Enable signaling child reaper from parent ns.
Posted by Dave Hansen on Tue, 29 May 2007 16:47:19 GMT
View Forum Message <> Reply to Message

On Fri, 2007-05-25 at 15:13 -0500, Serge E. Hallyn wrote:
>
> First note on that, (current == child_reaper(current)) is correct if we
> want to treat a container init as init no matter who sends the signal,
> however I contend that if a signal is sent from an ancester namespace,
> then we want to treat the task like any other task.
>
> Could I get some confirmation or rebuttal from Suka, Eric, Dave, or
> Pavel?

I agree with this. Signals should only be special when you're sending them "up the chain" to _your_ init process. We shouldn't do anything at all special to them when one process in a set of peers just happens to be the init for a child namespace.

-- Dave

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

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by Dave Hansen on Tue, 29 May 2007 16:54:32 GMT
View Forum Message <> Reply to Message

On Fri, 2007-05-25 at 13:44 -0700, sukadev@us.ibm.com wrote:
> Dave Hansen [hansendc@us.ibm.com] wrote:
> | On Thu, 2007-05-24 at 13:24 +0400, Pavel Emelianov wrote:
> | > > | > +int is_global_init(struct task_struct *tsk)
> | > > | > +{
> | > > | > + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);
> | > > |
> | > > | This can OOPS if you pass arbitrary task to this call...
> | > > | tsk->nsproxy can already be NULL.
> | > >
> | > > Hmm. You are right. btw, this could be a bisect issue. Patch 9 of uses
> | > > pid_ns from pid->upid_list and removes nsproxy->pid_ns.

> | >
> | > Yes, but that patch is not good either.
> | > task_pid(tsk) may become NULL as well and this will oops.
> |
> | Have you reviewed the call paths to make sure this can actually happen
> | in practice?
>
> task_pid() can be NULL when we are tearing down the task structure in
> release_task() and in the tiny window between detach_pid() and attach_pid()
> in de_thread().
>
> I think task_pid() is safe as long as it is called for 'current'. (we should
> probably add some comments)

If we only call it for "current", then perhaps we should just change the
function so that it doesn't take any arguments. That way nobody can
screw it up.

> I will double check my code, but I think all my calls to task_pid() and hence,
> to task_active_pid_ns() are safe, except for two cases:
>
> a) is_global_init(). There are a few calls to process other than
> current, but not sure if they are a problem.
>
> For instance in current code, unhandled_signal() checks
> tsk->pid == 1 and proceeds to derefernce tsk->sighand.
>
> If task_pid() is NULL because the task was in release_task(),
> then so is tsk->sighand.

Really? Are there barriers or locks to make this happen? Can you be
sure that compiler or cpu re-ordered code will keep this true?

> b) the temporary check I added in check_kill_permissions().
> (I need to address Serge's comment here anyway).
>
> To make is_global_init() more efficient and independent of task_pid(),
> can we steal a bit from task_struct->flags ? Like PF_KSWAPD, and there
> are unused bits :-)

It feels to me like we're adding too many hacks on hacks here. Let's
define the problem, because it sounds to me like we don't even know what
it really is. What is the problem here, again?

-- Dave

Subject: Re: [RFC][PATCH 0/16] Enable cloning of pid namespace
Posted by Badari Pulavarty on Tue, 29 May 2007 17:12:33 GMT
View Forum Message <> Reply to Message

On Tue, 2007-05-29 at 08:48 -0700, Dave Hansen wrote:
> On Tue, 2007-05-29 at 14:29 +0400, Pavel Emelianov wrote:
> > sukadev@us.ibm.com wrote:
> > > Dave, Serge,
> > >
> > > Here is my current pid namespace patchset. There are still a couple of
> > > issues I am investigating, but appreciate any feedback.
> > >
> > > Suka
> > >
> >
> > I tried to compile your patches with CONFIG_PID_NS=n and got this:
> >
> >   CC   kernel/configs.o
> > kernel/pid.c:361: warning: function declaration isn't a prototype
> > kernel/pid.c: In function `dup_struct_pid':
> > kernel/pid.c:489: warning: initialization makes pointer from integer without a cast
>
> Could you send along your actual .config?

Not needed :(

Index: linux-2.6.21/kernel/pid.c
===
--- linux-2.6.21.orig/kernel/pid.c 2007-05-25 09:06:30.000000000 -0700
+++ linux-2.6.21/kernel/pid.c 2007-05-29 10:07:39.000000000 -0700
@@ -357,7 +357,7 @@

 #else

-static int alloc_pid_ns()
+static struct pid_namespace *alloc_pid_ns(void)
 {
  static int warned;

@@ -365,7 +365,7 @@
   printk(KERN_INFO "WARNING: CLONE_NEWPID disabled\n");
   warned = 1;
  }
- return 0;

+ return NULL;
 }

 void zap_pid_ns_processes(struct pid_namespace *pid_ns)

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

---

Subject: Re: [RFC][PATCH 06/16] Define is_global_init()
Posted by Sukadev Bhattiprolu on Wed, 30 May 2007 00:29:27 GMT
View Forum Message <> Reply to Message

Dave Hansen [hansendc@us.ibm.com] wrote:
| On Fri, 2007-05-25 at 13:44 -0700, sukadev@us.ibm.com wrote:
| > Dave Hansen [hansendc@us.ibm.com] wrote:
| > | On Thu, 2007-05-24 at 13:24 +0400, Pavel Emelianov wrote:
| > | > > | > +int is_global_init(struct task_struct *tsk)
| > | > > | > +{
| > | > > | > + return (task_active_pid_ns(tsk) == &init_pid_ns && tsk->pid == 1);
| > | > > |
| > | > > | This can OOPS if you pass arbitrary task to this call...
| > | > > | tsk->nsproxy can already be NULL.
| > | > >
| > | > > Hmm. You are right. btw, this could be a bisect issue. Patch 9 of uses
| > | > > pid_ns from pid->upid_list and removes nsproxy->pid_ns.
| > | >
| > | > Yes, but that patch is not good either.
| > | > task_pid(tsk) may become NULL as well and this will oops.
| > |
| > | Have you reviewed the call paths to make sure this can actually happen
| > | in practice?
| >
| > task_pid() can be NULL when we are tearing down the task structure in
| > release_task() and in the tiny window between detach_pid() and attach_pid()
| > in de_thread().
| >
| > I think task_pid() is safe as long as it is called for 'current'. (we should
| > probably add some comments)
|
| If we only call it for "current", then perhaps we should just change the
| function so that it doesn't take any arguments. That way nobody can
| screw it up.

Well, if the caller can confirm that the tsk passed in to task_pid() is

not exiting, then it is ok to use. We have one such usage in do_fork()
where we use the task struct we just initialized. The task has not yet
been woken up.

|
| > I will double check my code, but I think all my calls to task_pid() and hence,
| > to task_active_pid_ns() are safe, except for two cases:
| >
| >   a) is_global_init(). There are a few calls to process other than
| >   current, but not sure if they are a problem.
| >
| >   For instance in current code, unhandled_signal() checks
| >   tsk->pid == 1 and proceeds to derefernce tsk->sighand.
| >
| >   If task_pid() is NULL because the task was in release_task(),
| >   then so is tsk->sighand.
|
| Really? Are there barriers or locks to make this happen? Can you be
| sure that compiler or cpu re-ordered code will keep this true?

I don't understand. Here is unhandled_signal() from 2.6.21-mm2.

```
int unhandled_signal(struct task_struct *tsk, int sig)
{
        if (is_init(tsk))
                return 1;
        if (tsk->ptrace & PT_PTRACED)
                return 0;
        return (tsk->sighand->action[sig-1].sa.sa_handler == SIG_IGN) ||
                (tsk->sighand->action[sig-1].sa.sa_handler == SIG_DFL);
}
```

My patch changed the is_init() to is_global_init() but the theory is
that is_global_init() is not safe since it uses task_pid() which
can return NULL if @tsk is exiting.

But if @tsk is exiting, tsk->sighand will also be NULL. So either
the current callers have somehow ensured @tsk is not exiting or
they are risking accessing tsk->sighand.

Not sure how compiler/cpu-reordering changes things. Can you elaborate ?

|
| >   b) the temporary check I added in check_kill_permissions().
| >   (I need to address Serge's comment here anyway).
| >
| > To make is_global_init() more efficient and independent of task_pid(),
| > can we steal a bit from task_struct->flags ? Like PF_KSWAPD, and there

| > are unused bits :-)
|
| It feels to me like we're adding too many hacks on hacks here. Let's
| define the problem, because it sounds to me like we don't even know what
| it really is. What is the problem here, again?

We need an interface is_global_init() that tells us if the given process
is /sbin/init (which is the process with pid_t == 1 in init_pid_ns).

Why we need is_global_init(): to allow/deny some facilities (eg: you
cannot send arbitrary signals to the process).

How to implement is_global_init(): One simple/efficient way is to use a
bit in task->flags.

Another way is to:

 - ensure tsk->pid == 1 and
 - active pid ns of @tsk is init_pid_ns.

To check active pid ns of a process we currently need task_pid() which
may not be safe for all processes at all times.

A releated interface is is_container_init() which can be defined as
the process with pid_t == 1 in its active pid namespace.
|
| -- Dave

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

---

Subject: Re: [RFC][PATCH 14/16] Introduce proc_mnt for pid_ns
Posted by Pavel Emelianov on Thu, 31 May 2007 11:48:21 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Subject: Introduce proc_mnt for pid_ns
>
> From: Dave Hansen <hansendc@us.ibm.com>
>
> The following patch completes the removal of the global proc_mnt.
> It fetches the mnt on which to do dentry invalidations from the
> pid_namespace in which the task appears.
>
> For now, there is only one pid namespace in mainline so this is
> straightforward. In the -lxc tree we'll have to do something

> more complex. The proc_flush_task() code takes a task, and
> needs to be able to find the corresponding proc superblocks on
> which that tasks's /proc/<pid> directories could appear. We
> can tell in which pid namespaces a task appears, so I put a
> pointer from the pid namespace to the corresponding proc_mnt.
>
> /proc currently has some special code to make sure that the root
> directory gets set up correctly. It proc_mnt variable in order
> to find its way to the root inode.
>
> Signed-off-by: Dave Hansen <haveblue@us.ibm.com>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
>
> fs/proc/base.c | 32 +++++++++++++++++++++++-
> fs/proc/inode.c | 11 +++++++-
> fs/proc/root.c | 52 +++++++++++++++++++++++++++--------------
> include/linux/pid_namespace.h | 1
> include/linux/proc_fs.h | 1
> 5 files changed, 75 insertions(+), 22 deletions(-)
>

[snip]

> @@ -2071,6 +2084,21 @@ out:
>   return;
> }
>
> +void proc_flush_task(struct task_struct *task)
> +{
> + int i;
> + struct pid *pid;
> + struct upid* upid;
> +
> + pid = task_pid(task);
> + if (!pid)
> + return;

The code below will never be called as task flushes all his pids
in __unhash_process() that happens before this. Or did I miss smth?

> + upid = &pid->upid_list[0];
> + for (i = 0; i < pid->num_upids; i++, upid++)
> + proc_flush_task_from_pid_ns(task, upid->pid_ns);
> +}
> +
> static struct dentry *proc_pid_instantiate(struct inode *dir,
>   struct dentry * dentry,

> struct task_struct *task, const void *ptr)

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

---

Subject: Re: [RFC][PATCH 14/16] Introduce proc_mnt for pid_ns
Posted by Dave Hansen on Thu, 31 May 2007 16:40:04 GMT
View Forum Message <> Reply to Message

On Thu, 2007-05-31 at 15:48 +0400, Pavel Emelianov wrote:
> > +void proc_flush_task(struct task_struct *task)
> > +{
> > + int i;
> > + struct pid *pid;
> > + struct upid* upid;
> > +
> > + pid = task_pid(task);
> > + if (!pid)
> > + return;
>
> The code below will never be called as task flushes all his pids
> in __unhash_process() that happens before this. Or did I miss smth?

Nope, that's a very nice catch. Suka is working on a patch to fix this right now. We just need to keep the list of pids accessible a little longer, but I think Suka ran into a chicken-and-egg problem while trying to solve this. Suka, care to post your workaround?

-- Dave

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

---

Subject: Re: [RFC][PATCH 14/16] Introduce proc_mnt for pid_ns
Posted by Sukadev Bhattiprolu on Thu, 31 May 2007 19:50:07 GMT
View Forum Message <> Reply to Message

Dave Hansen [hansendc@us.ibm.com] wrote:
| On Thu, 2007-05-31 at 15:48 +0400, Pavel Emelianov wrote:
| > > +void proc_flush_task(struct task_struct *task)
| > > +{
| > > + int i;

| > > +	struct pid *pid;
| > > +	struct upid* upid;
| > > +
| > > +	pid = task_pid(task);
| > > +	if (!pid)
| > > +		return;
| >
| > The code below will never be called as task flushes all his pids
| > in __unhash_process() that happens before this. Or did I miss smth?
|
| Nope, that's a very nice catch. Suka is working on a patch to fix this
| right now. We just need to keep the list of pids accessible a little
| longer, but I think Suka ran into a chicken-and-egg problem while trying
| to solve this. Suka, care to post your workaround?

Yes. the chicken and egg problem is in the order of the __unhash_process()
and proc_flush_task().

With __unhash_process() first, proc_flush_task() has no way of knowing
the pid namespaces the process belongs to. With proc_flush_task() first,
we find the process but have its dentry removed from proc. This crashes
when I run a test in a tight loop.

A quick/dirty fix is to save the pid namespace list before __unhash_process()
and use that in proc_flush_task(). That would make the exit() path quite
expensive. I am still investigating the crash and looking for a better option.

|
| -- Dave

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