Subject: [RFC][PATCH 0/4] Devpts namespace
Posted by Sukadev Bhattiprolu on Wed, 06 Feb 2008 05:04:28 GMT
View Forum Message <> Reply to Message

Serge, Matt, please sign-off on these patches as you see fit.

Devpts namespace patchset

In continuation of the implementation of containers in mainline, we need to support multiple PTY namespaces so that the PTY index (ie the tty names) in one container is independent of the PTY indices of other containers. For instance this would allow each container to have a '/dev/pts/0' PTY and refer to different terminals.

[PATCH 1/4]: Factor out PTY index allocation
[PATCH 2/4]: Use interface to access allocated_ptys
[PATCH 3/4]: Enable multiple mounts of /dev/pts
[PATCH 4/4]: Enable cloning PTY namespaces

Todo:

- This patchset depends on availability of additional clone flags !!!
- Needs more testing.

Changelog:

This patchset is based on earlier versions developed by Serge Hallyn and Matt Helsley.

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

Subject: [RFC][PATCH 1/4]: Factor out PTY index allocation
Posted by Sukadev Bhattiprolu on Wed, 06 Feb 2008 05:09:46 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [RFC][PATCH 1/4]: Factor out PTY index allocation

Factor out the code used to allocate/free a pts index into new interfaces, devpts_new_index() and devpts_kill_index(). This localizes the external data structures used in managing the pts indices.

```
Changelog:
 - Version 0: Based on earlier versions from Serge Hallyn and
   Matt Helsley.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 drivers/char/tty_io.c     |   40 ++++++----------------------------
 fs/devpts/inode.c         |   42 ++++++++++++++++++++++++++++++++++++++-
 include/linux/devpts_fs.h |    4 ++++
 3 files changed, 51 insertions(+), 35 deletions(-)

Index: linux-2.6.24/drivers/char/tty_io.c
===================================================================
--- linux-2.6.24.orig/drivers/char/tty_io.c 2008-01-24 14:58:37.000000000 -0800
+++ linux-2.6.24/drivers/char/tty_io.c 2008-02-05 17:17:11.000000000 -0800
@@ -90,7 +90,6 @@
 #include <linux/module.h>
 #include <linux/smp_lock.h>
 #include <linux/device.h>
-#include <linux/idr.h>
 #include <linux/wait.h>
 #include <linux/bitops.h>
 #include <linux/delay.h>
@@ -136,9 +135,6 @@ EXPORT_SYMBOL(tty_mutex);

 #ifdef CONFIG_UNIX98_PTYS
 extern struct tty_driver *ptm_driver; /* Unix98 pty masters; for /dev/ptmx */
-extern int pty_limit;  /* Config limit on Unix98 ptys */
-static DEFINE_IDR(allocated_ptys);
-static DECLARE_MUTEX(allocated_ptys_lock);
 static int ptmx_open(struct inode *, struct file *);
 #endif

@@ -2568,15 +2564,9 @@ static void release_dev(struct file * fi
   */
  release_tty(tty, idx);

-#ifdef CONFIG_UNIX98_PTYS
  /* Make this pty number available for reallocation */
- if (devpts) {
-  down(&allocated_ptys_lock);
-  idr_remove(&allocated_ptys, idx);
-  up(&allocated_ptys_lock);
- }
-#endif
-
+ if (devpts)
+  devpts_kill_index(idx);
```

```
 }

 /**
@@ -2732,29 +2722,13 @@ static int ptmx_open(struct inode * inod
  struct tty_struct *tty;
  int retval;
  int index;
- int idr_ret;

  nonseekable_open(inode, filp);

  /* find a device that is not in use. */
- down(&allocated_ptys_lock);
- if (!idr_pre_get(&allocated_ptys, GFP_KERNEL)) {
-  up(&allocated_ptys_lock);
-  return -ENOMEM;
- }
- idr_ret = idr_get_new(&allocated_ptys, NULL, &index);
- if (idr_ret < 0) {
-  up(&allocated_ptys_lock);
-  if (idr_ret == -EAGAIN)
-   return -ENOMEM;
-  return -EIO;
- }
- if (index >= pty_limit) {
-  idr_remove(&allocated_ptys, index);
-  up(&allocated_ptys_lock);
-  return -EIO;
- }
- up(&allocated_ptys_lock);
+ index = devpts_new_index();
+ if (index < 0)
+  return index;

  mutex_lock(&tty_mutex);
  retval = init_dev(ptm_driver, index, &tty);
@@ -2781,9 +2755,7 @@ out1:
  release_dev(filp);
  return retval;
 out:
- down(&allocated_ptys_lock);
- idr_remove(&allocated_ptys, index);
- up(&allocated_ptys_lock);
+ devpts_kill_index(index);
  return retval;
 }
 #endif
Index: linux-2.6.24/fs/devpts/inode.c
```

```
==========================================================================
--- linux-2.6.24.orig/fs/devpts/inode.c 2008-01-24 14:58:37.000000000 -0800
+++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 17:17:11.000000000 -0800
@@ -17,12 +17,17 @@
 #include <linux/namei.h>
 #include <linux/mount.h>
 #include <linux/tty.h>
+#include <linux/idr.h>
 #include <linux/devpts_fs.h>
 #include <linux/parser.h>
 #include <linux/fsnotify.h>

 #define DEVPTS_SUPER_MAGIC 0x1cd1

+extern int pty_limit;  /* Config limit on Unix98 ptys */
+static DEFINE_IDR(allocated_ptys);
+static DECLARE_MUTEX(allocated_ptys_lock);
+
 static struct vfsmount *devpts_mnt;
 static struct dentry *devpts_root;

@@ -156,9 +161,44 @@ static struct dentry *get_node(int num)
  return lookup_one_len(s, root, sprintf(s, "%d", num));
 }

+int devpts_new_index(void)
+{
+ int index;
+ int idr_ret;
+
+retry:
+ if (!idr_pre_get(&allocated_ptys, GFP_KERNEL)) {
+  return -ENOMEM;
+ }
+
+ down(&allocated_ptys_lock);
+ idr_ret = idr_get_new(&allocated_ptys, NULL, &index);
+ if (idr_ret < 0) {
+  up(&allocated_ptys_lock);
+  if (idr_ret == -EAGAIN)
+   goto retry;
+  return -EIO;
+ }
+
+ if (index >= pty_limit) {
+  idr_remove(&allocated_ptys, index);
+  up(&allocated_ptys_lock);
+  return -EIO;
```

```
+ }
+ up(&allocated_ptys_lock);
+ return index;
+}
+
+void devpts_kill_index(int idx)
+{
+ down(&allocated_ptys_lock);
+ idr_remove(&allocated_ptys, idx);
+ up(&allocated_ptys_lock);
+}
+
 int devpts_pty_new(struct tty_struct *tty)
 {
- int number = tty->index;
+ int number = tty->index; /* tty layer puts index from devpts_new_index() in here */
  struct tty_driver *driver = tty->driver;
  dev_t device = MKDEV(driver->major, driver->minor_start+number);
  struct dentry *dentry;
Index: linux-2.6.24/include/linux/devpts_fs.h
===================================================================
--- linux-2.6.24.orig/include/linux/devpts_fs.h 2008-01-24 14:58:37.000000000 -0800
+++ linux-2.6.24/include/linux/devpts_fs.h 2008-02-05 17:17:11.000000000 -0800
@@ -17,6 +17,8 @@

 #ifdef CONFIG_UNIX98_PTYS

+int devpts_new_index(void);
+void devpts_kill_index(int idx);
 int devpts_pty_new(struct tty_struct *tty);      /* mknod in devpts */
 struct tty_struct *devpts_get_tty(int number);  /* get tty structure */
 void devpts_pty_kill(int number);   /* unlink */
@@ -24,6 +26,8 @@ void devpts_pty_kill(int number);   /* u
 #else

 /* Dummy stubs in the no-pty case */
+static inline int devpts_new_index(void) { return -EINVAL; }
+static inline void devpts_kill_index(int idx) { }
 static inline int devpts_pty_new(struct tty_struct *tty) { return -EINVAL; }
 static inline struct tty_struct *devpts_get_tty(int number) { return NULL; }
 static inline void devpts_pty_kill(int number) { }
```

Subject: [RFC][PATCH 2/4]: Use interface to access allocated_ptys
Posted by Sukadev Bhattiprolu on Wed, 06 Feb 2008 05:10:27 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [RFC][PATCH 2/4]: Use interface to access allocated_ptys

In preparation for supporting multiple PTY namespaces, use an inline
function to access the 'allocated_ptys' idr.

Changelog:
 - Version 0: Based on earlier versions from Serge Hallyn and
   Matt Helsley.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Acked-by: Serge Hallyn <serue@us.ibm.com>
---
 fs/devpts/inode.c | 13 +++++++++----
 1 file changed, 9 insertions(+), 4 deletions(-)

Index: linux-2.6.24/fs/devpts/inode.c
===
--- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 17:17:11.000000000 -0800
+++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 17:30:52.000000000 -0800
@@ -28,6 +28,11 @@ extern int pty_limit; /* Config limit o
 static DEFINE_IDR(allocated_ptys);
 static DECLARE_MUTEX(allocated_ptys_lock);

+static inline struct idr *current_pts_ns_allocated_ptys(void)
+{
+ return &allocated_ptys;
+}
+
 static struct vfsmount *devpts_mnt;
 static struct dentry *devpts_root;

@@ -167,12 +172,12 @@ int devpts_new_index(void)
  int idr_ret;

 retry:
- if (!idr_pre_get(&allocated_ptys, GFP_KERNEL)) {
+ if (!idr_pre_get(current_pts_ns_allocated_ptys(), GFP_KERNEL)) {
   return -ENOMEM;
  }

  down(&allocated_ptys_lock);
- idr_ret = idr_get_new(&allocated_ptys, NULL, &index);
+ idr_ret = idr_get_new(current_pts_ns_allocated_ptys(), NULL, &index);
  if (idr_ret < 0) {

```
   up(&allocated_ptys_lock);
   if (idr_ret == -EAGAIN)
@@ -181,7 +186,7 @@ retry:
  }

  if (index >= pty_limit) {
-  idr_remove(&allocated_ptys, index);
+  idr_remove(current_pts_ns_allocated_ptys(), index);
   up(&allocated_ptys_lock);
   return -EIO;
  }
@@ -192,7 +197,7 @@ retry:
 void devpts_kill_index(int idx)
 {
  down(&allocated_ptys_lock);
- idr_remove(&allocated_ptys, idx);
+ idr_remove(current_pts_ns_allocated_ptys(), idx);
  up(&allocated_ptys_lock);
 }
```

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

---

Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Sukadev Bhattiprolu on Wed, 06 Feb 2008 05:10:55 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts

To support multiple PTY namespaces, we should be allow multiple mounts of
/dev/pts, once within each PTY namespace.

This patch removes the get_sb_single() in devpts_get_sb() and uses test and
set sb interfaces to allow remounting /dev/pts. The patch also removes the
globals, 'devpts_root' and uses current_pts_mnt() to access 'devpts_mnt'

Changelog:
 - Version 0: Based on earlier versions from Serge Hallyn and
   Matt Helsley.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
---
 fs/devpts/inode.c | 120 ++---------
 1 file changed, 101 insertions(+), 19 deletions(-)

```
Index: linux-2.6.24/fs/devpts/inode.c
===================================================================
--- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 17:30:52.000000000 -0800
+++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
@@ -34,7 +34,10 @@ static inline struct idr *current_pts_ns
 }

 static struct vfsmount *devpts_mnt;
-static struct dentry *devpts_root;
+static inline struct vfsmount *current_pts_ns_mnt(void)
+{
+ return devpts_mnt;
+}

 static struct {
  int setuid;
@@ -130,7 +133,7 @@ devpts_fill_super(struct super_block *s,
  inode->i_fop = &simple_dir_operations;
  inode->i_nlink = 2;

- devpts_root = s->s_root = d_alloc_root(inode);
+ s->s_root = d_alloc_root(inode);
  if (s->s_root)
   return 0;

@@ -140,10 +143,53 @@ fail:
  return -ENOMEM;
 }

+/*
+ * We use test and set super-block operations to help determine whether we
+ * need a new super-block for this namespace. get_sb() walks the list of
+ * existing devpts supers, comparing them with the @data ptr. Since we
+ * passed 'current's namespace as the @data pointer we can compare the
+ * namespace pointer in the super-block's 's_fs_info'.  If the test is
+ * TRUE then get_sb() returns a new active reference to the super block.
+ * Otherwise, it helps us build an active reference to a new one.
+ */
+
+static int devpts_test_sb(struct super_block *sb, void *data)
+{
+ return sb->s_fs_info == data;
+}
+
+static int devpts_set_sb(struct super_block *sb, void *data)
+{
+ sb->s_fs_info = data;
```

```
+ return set_anon_super(sb, NULL);
+}
+
 static int devpts_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
 {
- return get_sb_single(fs_type, flags, data, devpts_fill_super, mnt);
+ struct super_block *sb;
+ int err;
+
+ /* hereafter we're very simlar to get_sb_nodev */
+ sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
+ if (IS_ERR(sb))
+  return PTR_ERR(sb);
+
+ if (sb->s_root)
+  return simple_set_mnt(mnt, sb);
+
+ sb->s_flags = flags;
+ err = devpts_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
+ if (err) {
+  up_write(&sb->s_umount);
+  deactivate_super(sb);
+  return err;
+ }
+
+ sb->s_flags |= MS_ACTIVE;
+ devpts_mnt = mnt;
+
+ return simple_set_mnt(mnt, sb);
 }

 static struct file_system_type devpts_fs_type = {
@@ -158,10 +204,9 @@ static struct file_system_type devpts_fs
  * to the System V naming convention
  */

-static struct dentry *get_node(int num)
+static struct dentry *get_node(struct dentry *root, int num)
 {
  char s[12];
- struct dentry *root = devpts_root;
  mutex_lock(&root->d_inode->i_mutex);
  return lookup_one_len(s, root, sprintf(s, "%d", num));
 }
@@ -207,12 +252,28 @@ int devpts_pty_new(struct tty_struct *tt
  struct tty_driver *driver = tty->driver;
  dev_t device = MKDEV(driver->major, driver->minor_start+number);
```

```
  struct dentry *dentry;
- struct inode *inode = new_inode(devpts_mnt->mnt_sb);
+ struct dentry *root;
+ struct vfsmount *mnt;
+ struct inode *inode;
+

  /* We're supposed to be given the slave end of a pty */
  BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
  BUG_ON(driver->subtype != PTY_TYPE_SLAVE);

+ mnt = current_pts_ns_mnt();
+ if (!mnt)
+  return -ENOSYS;
+ root = mnt->mnt_root;
+
+ mutex_lock(&root->d_inode->i_mutex);
+ inode = idr_find(current_pts_ns_allocated_ptys(), number);
+ mutex_unlock(&root->d_inode->i_mutex);
+
+ if (inode && !IS_ERR(inode))
+  return -EEXIST;
+
+ inode = new_inode(mnt->mnt_sb);
  if (!inode)
   return -ENOMEM;

@@ -222,23 +283,31 @@ int devpts_pty_new(struct tty_struct *tt
  inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
  init_special_inode(inode, S_IFCHR|config.mode, device);
  inode->i_private = tty;
+ idr_replace(current_pts_ns_allocated_ptys(), inode, number);

- dentry = get_node(number);
+ dentry = get_node(root, number);
  if (!IS_ERR(dentry) && !dentry->d_inode) {
   d_instantiate(dentry, inode);
-  fsnotify_create(devpts_root->d_inode, dentry);
+  fsnotify_create(root->d_inode, dentry);
  }

- mutex_unlock(&devpts_root->d_inode->i_mutex);
+ mutex_unlock(&root->d_inode->i_mutex);

  return 0;
 }

 struct tty_struct *devpts_get_tty(int number)
```

```
 {
- struct dentry *dentry = get_node(number);
+ struct vfsmount *mnt;
+ struct dentry *dentry;
  struct tty_struct *tty;

+ mnt = current_pts_ns_mnt();
+ if (!mnt)
+  return NULL;
+
+ dentry = get_node(mnt->mnt_root, number);
+
  tty = NULL;
  if (!IS_ERR(dentry)) {
   if (dentry->d_inode)
@@ -246,14 +315,21 @@ struct tty_struct *devpts_get_tty(int nu
   dput(dentry);
  }

- mutex_unlock(&devpts_root->d_inode->i_mutex);
+ mutex_unlock(&mnt->mnt_root->d_inode->i_mutex);

  return tty;
 }

 void devpts_pty_kill(int number)
 {
- struct dentry *dentry = get_node(number);
+ struct dentry *dentry;
+ struct dentry *root;
+ struct vfsmount *mnt;
+
+ mnt = current_pts_ns_mnt();
+ root = mnt->mnt_root;
+
+ dentry = get_node(root, number);

  if (!IS_ERR(dentry)) {
   struct inode *inode = dentry->d_inode;
@@ -264,17 +340,23 @@ void devpts_pty_kill(int number)
   }
   dput(dentry);
  }
- mutex_unlock(&devpts_root->d_inode->i_mutex);
+ mutex_unlock(&root->d_inode->i_mutex);
 }

 static int __init init_devpts_fs(void)
```

```
 {
- int err = register_filesystem(&devpts_fs_type);
- if (!err) {
-  devpts_mnt = kern_mount(&devpts_fs_type);
-  if (IS_ERR(devpts_mnt))
-   err = PTR_ERR(devpts_mnt);
- }
+ struct vfsmount *mnt;
+ int err;
+
+ err = register_filesystem(&devpts_fs_type);
+ if (err)
+  return err;
+
+ mnt = kern_mount_data(&devpts_fs_type, NULL);
+ if (IS_ERR(mnt))
+  err = PTR_ERR(mnt);
+ else
+  devpts_mnt = mnt;
  return err;
 }
```

_____

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

---

Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
Posted by Sukadev Bhattiprolu on Wed, 06 Feb 2008 05:11:17 GMT
View Forum Message <> Reply to Message

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces

Enable cloning PTY namespaces.

TODO:
 This version temporarily uses the clone flag '0x80000000' which
 is unused in mainline atm, but used for CLONE_IO in -mm.
 While we must extend clone() (urgently) to solve this, it hopefully
 does not affect review of the rest of this patchset.

Changelog:
 - Version 0: Based on earlier versions from Serge Hallyn and
   Matt Helsley.

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

```
---
 fs/devpts/inode.c         |   84 +++++++++++++++++++++++++++++++++++++-------
 include/linux/devpts_fs.h |   52 ++++++++++++++++++++++++++++
 include/linux/init_task.h |    1
 include/linux/nsproxy.h   |    2 +
 include/linux/sched.h     |    2 +
 kernel/fork.c             |    2 -
 kernel/nsproxy.c          |   17 ++++++++-
 7 files changed, 146 insertions(+), 14 deletions(-)

Index: linux-2.6.24/fs/devpts/inode.c
===================================================================
--- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
+++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 20:27:41.000000000 -0800
@@ -25,18 +25,25 @@
 #define DEVPTS_SUPER_MAGIC 0x1cd1

 extern int pty_limit;  /* Config limit on Unix98 ptys */
-static DEFINE_IDR(allocated_ptys);
 static DECLARE_MUTEX(allocated_ptys_lock);
+static struct file_system_type devpts_fs_type;
+
+struct pts_namespace init_pts_ns = {
+ .kref = {
+  .refcount = ATOMIC_INIT(2),
+ },
+ .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
+ .mnt = NULL,
+};

 static inline struct idr *current_pts_ns_allocated_ptys(void)
 {
- return &allocated_ptys;
+ return &current->nsproxy->pts_ns->allocated_ptys;
 }

-static struct vfsmount *devpts_mnt;
 static inline struct vfsmount *current_pts_ns_mnt(void)
 {
- return devpts_mnt;
+ return current->nsproxy->pts_ns->mnt;
 }

 static struct {
@@ -59,6 +66,42 @@ static match_table_t tokens = {
  {Opt_err, NULL}
 };
```

```
+struct pts_namespace *new_pts_ns(void)
+{
+ struct pts_namespace *ns;
+
+ ns = kmalloc(sizeof(*ns), GFP_KERNEL);
+ if (!ns)
+  return ERR_PTR(-ENOMEM);
+
+ ns->mnt = kern_mount_data(&devpts_fs_type, ns);
+ if (IS_ERR(ns->mnt)) {
+  kfree(ns);
+  return ERR_PTR(PTR_ERR(ns->mnt));
+ }
+
+ idr_init(&ns->allocated_ptys);
+ kref_init(&ns->kref);
+
+ return ns;
+}
+
+void free_pts_ns(struct kref *ns_kref)
+{
+ struct pts_namespace *ns;
+
+ ns = container_of(ns_kref, struct pts_namespace, kref);
+ BUG_ON(ns == &init_pts_ns);
+
+ mntput(ns->mnt);
+ /*
+  * TODO:
+  *      idr_remove_all(&ns->allocated_ptys); introduced in 2.6.23
+  */
+ idr_destroy(&ns->allocated_ptys);
+ kfree(ns);
+}
+
 static int devpts_remount(struct super_block *sb, int *flags, char *data)
 {
  char *p;
@@ -160,18 +203,27 @@ static int devpts_test_sb(struct super_b

 static int devpts_set_sb(struct super_block *sb, void *data)
 {
- sb->s_fs_info = data;
+ struct pts_namespace *ns = data;
+
+ sb->s_fs_info = get_pts_ns(ns);
  return set_anon_super(sb, NULL);
```

```
 }

 static int devpts_get_sb(struct file_system_type *fs_type,
  int flags, const char *dev_name, void *data, struct vfsmount *mnt)
 {
+ struct pts_namespace *ns;
  struct super_block *sb;
  int err;

+ /* hereafter we're very similar to proc_get_sb */
+ if (flags & MS_KERNMOUNT)
+  ns = data;
+ else
+  ns = current->nsproxy->pts_ns;
+
  /* hereafter we're very simlar to get_sb_nodev */
- sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
+ sb = sget(fs_type, devpts_test_sb, devpts_set_sb, ns);
  if (IS_ERR(sb))
   return PTR_ERR(sb);

@@ -187,16 +239,25 @@ static int devpts_get_sb(struct file_sys
  }

  sb->s_flags |= MS_ACTIVE;
- devpts_mnt = mnt;
+ ns->mnt = mnt;

  return simple_set_mnt(mnt, sb);
 }

+static void devpts_kill_sb(struct super_block *sb)
+{
+       struct pts_namespace *ns;
+
+       ns = sb->s_fs_info;
+       kill_anon_super(sb);
+       put_pts_ns(ns);
+}
+
 static struct file_system_type devpts_fs_type = {
  .owner  = THIS_MODULE,
  .name  = "devpts",
  .get_sb  = devpts_get_sb,
- .kill_sb = kill_anon_super,
+ .kill_sb = devpts_kill_sb,
 };
```

```
 /*
@@ -352,18 +413,19 @@ static int __init init_devpts_fs(void)
  if (err)
   return err;

- mnt = kern_mount_data(&devpts_fs_type, NULL);
+ mnt = kern_mount_data(&devpts_fs_type, &init_pts_ns);
  if (IS_ERR(mnt))
   err = PTR_ERR(mnt);
  else
-  devpts_mnt = mnt;
+  init_pts_ns.mnt = mnt;
  return err;
 }

 static void __exit exit_devpts_fs(void)
 {
  unregister_filesystem(&devpts_fs_type);
- mntput(devpts_mnt);
+ mntput(init_pts_ns.mnt);
+ init_pts_ns.mnt = NULL;
 }

 module_init(init_devpts_fs)
Index: linux-2.6.24/include/linux/devpts_fs.h
===================================================================
--- linux-2.6.24.orig/include/linux/devpts_fs.h 2008-02-05 19:16:39.000000000 -0800
+++ linux-2.6.24/include/linux/devpts_fs.h 2008-02-05 20:21:08.000000000 -0800
@@ -14,9 +14,45 @@
 #define _LINUX_DEVPTS_FS_H

 #include <linux/errno.h>
+#include <linux/nsproxy.h>
+#include <linux/kref.h>
+#include <linux/idr.h>
+
+struct pts_namespace {
+ struct kref kref;
+ struct idr allocated_ptys;
+ struct vfsmount *mnt;
+};
+
+extern struct pts_namespace init_pts_ns;

 #ifdef CONFIG_UNIX98_PTYS

+extern struct pts_namespace *new_pts_ns(void);
+extern void free_pts_ns(struct kref *kref);
```

```
+
+static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
+{
+ if (ns)
+  kref_get(&ns->kref);
+ return ns;
+}
+
+static inline void put_pts_ns(struct pts_namespace *ns)
+{
+ if (ns)
+  kref_put(&ns->kref, free_pts_ns);
+}
+
+static inline struct pts_namespace *copy_pts_ns(unsigned long flags,
+  struct pts_namespace *old_ns)
+{
+ if (flags & CLONE_NEWPTS)
+  return new_pts_ns();
+ else
+  return get_pts_ns(old_ns);
+}
+
 int devpts_new_index(void);
 void devpts_kill_index(int idx);
 int devpts_pty_new(struct tty_struct *tty);      /* mknod in devpts */
@@ -26,6 +62,22 @@ void devpts_pty_kill(int number);   /* u
 #else

 /* Dummy stubs in the no-pty case */
+
+static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
+{
+ return &init_pts_ns;
+}
+
+static inline void put_pts_ns(struct pts_namespace *ns) { }
+
+static inline struct pts_namespace *copy_pts_ns(unsigned long flags,
+  struct pts_namespace *old_ns)
+{
+ if (flags & CLONE_NEWPTS)
+  return ERR_PTR(-EINVAL);
+ return old_ns;
+}
+
 static inline int devpts_new_index(void) { return -EINVAL; }
 static inline void devpts_kill_index(int idx) { }
```

```
 static inline int devpts_pty_new(struct tty_struct *tty) { return -EINVAL; }
Index: linux-2.6.24/include/linux/init_task.h
===================================================================
--- linux-2.6.24.orig/include/linux/init_task.h 2008-02-05 19:16:39.000000000 -0800
+++ linux-2.6.24/include/linux/init_task.h 2008-02-05 19:18:00.000000000 -0800
@@ -77,6 +77,7 @@ extern struct nsproxy init_nsproxy;
  .mnt_ns  = NULL,      \
  INIT_NET_NS(net_ns)                                     \
  INIT_IPC_NS(ipc_ns)      \
+ .pts_ns  = &init_pts_ns,     \
  .user_ns = &init_user_ns,    \
 }

Index: linux-2.6.24/include/linux/nsproxy.h
===================================================================
--- linux-2.6.24.orig/include/linux/nsproxy.h 2008-02-05 19:16:39.000000000 -0800
+++ linux-2.6.24/include/linux/nsproxy.h 2008-02-05 19:18:00.000000000 -0800
@@ -8,6 +8,7 @@ struct mnt_namespace;
 struct uts_namespace;
 struct ipc_namespace;
 struct pid_namespace;
+struct pts_namespace;

 /*
  * A structure to contain pointers to all per-process
@@ -29,6 +30,7 @@ struct nsproxy {
  struct pid_namespace *pid_ns;
  struct user_namespace *user_ns;
  struct net       *net_ns;
+ struct pts_namespace *pts_ns;
 };
 extern struct nsproxy init_nsproxy;

Index: linux-2.6.24/include/linux/sched.h
===================================================================
--- linux-2.6.24.orig/include/linux/sched.h 2008-02-05 19:16:39.000000000 -0800
+++ linux-2.6.24/include/linux/sched.h 2008-02-05 19:54:05.000000000 -0800
@@ -27,6 +27,8 @@
 #define CLONE_NEWUSER  0x10000000 /* New user namespace */
 #define CLONE_NEWPID  0x20000000 /* New pid namespace */
 #define CLONE_NEWNET  0x40000000 /* New network namespace */
+#define CLONE_NEWPTS     (CLONE_NEWNS|0x80000000) /* Temporary - only for patch
review */
+       /* Badly need to /extend clone() !!! */

 /*
  * Scheduling policies
Index: linux-2.6.24/kernel/fork.c
```

```
===================================================================
--- linux-2.6.24.orig/kernel/fork.c 2008-02-05 19:16:39.000000000 -0800
+++ linux-2.6.24/kernel/fork.c 2008-02-05 19:18:00.000000000 -0800
@@ -1655,7 +1655,7 @@ asmlinkage long sys_unshare(unsigned lon
  if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
     CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWUSER|
-    CLONE_NEWNET))
+    CLONE_NEWNET|CLONE_NEWPTS))
   goto bad_unshare_out;

  if ((err = unshare_thread(unshare_flags)))
Index: linux-2.6.24/kernel/nsproxy.c
===================================================================
--- linux-2.6.24.orig/kernel/nsproxy.c 2008-02-05 19:16:39.000000000 -0800
+++ linux-2.6.24/kernel/nsproxy.c 2008-02-05 19:18:00.000000000 -0800
@@ -21,6 +21,7 @@
 #include <linux/utsname.h>
 #include <linux/pid_namespace.h>
 #include <net/net_namespace.h>
+#include <linux/devpts_fs.h>

 static struct kmem_cache *nsproxy_cachep;

@@ -92,8 +93,17 @@ static struct nsproxy *create_new_namesp
   goto out_net;
  }

+ new_nsp->pts_ns = copy_pts_ns(flags, tsk->nsproxy->pts_ns);
+ if (IS_ERR(new_nsp->pts_ns)) {
+  err = PTR_ERR(new_nsp->pts_ns);
+  goto out_pts;
+ }
+
  return new_nsp;

+out_pts:
+ if (new_nsp->net_ns)
+  put_net(new_nsp->net_ns);
 out_net:
  if (new_nsp->user_ns)
   put_user_ns(new_nsp->user_ns);
@@ -130,7 +140,8 @@ int copy_namespaces(unsigned long flags,
  get_nsproxy(old_ns);

  if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
-    CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET)))
+    CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET |
```

```
+    CLONE_NEWPTS)))
   return 0;

  if (!capable(CAP_SYS_ADMIN)) {
@@ -169,6 +180,8 @@ void free_nsproxy(struct nsproxy *ns)
   put_pid_ns(ns->pid_ns);
  if (ns->user_ns)
   put_user_ns(ns->user_ns);
+ if (ns->pts_ns)
+  put_pts_ns(ns->pts_ns);
  put_net(ns->net_ns);
  kmem_cache_free(nsproxy_cachep, ns);
 }
@@ -183,7 +196,7 @@ int unshare_nsproxy_namespaces(unsigned
  int err = 0;

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

  if (!capable(CAP_SYS_ADMIN))
```

Subject: Re: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
Posted by Pavel Emelianov on Wed, 06 Feb 2008 09:04:15 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
>
> Enable cloning PTY namespaces.
>
> TODO:
> This version temporarily uses the clone flag '0x80000000' which
> is unused in mainline atm, but used for CLONE_IO in -mm.
> While we must extend clone() (urgently) to solve this, it hopefully
> does not affect review of the rest of this patchset.
>
> Changelog:
> - Version 0: Based on earlier versions from Serge Hallyn and
>   Matt Helsley.
>

> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> fs/devpts/inode.c | 84 ++++++++++++++++++++++++++++++++++++-------
> include/linux/devpts_fs.h | 52 ++++++++++++++++++++++++++++
> include/linux/init_task.h | 1
> include/linux/nsproxy.h | 2 +
> include/linux/sched.h | 2 +
> kernel/fork.c | 2 -
> kernel/nsproxy.c | 17 ++++++++-
> 7 files changed, 146 insertions(+), 14 deletions(-)
>
> Index: linux-2.6.24/fs/devpts/inode.c
> ===
> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 20:27:41.000000000 -0800
> @@ -25,18 +25,25 @@
> #define DEVPTS_SUPER_MAGIC 0x1cd1
>
> extern int pty_limit; /* Config limit on Unix98 ptys */
> -static DEFINE_IDR(allocated_ptys);
> static DECLARE_MUTEX(allocated_ptys_lock);
> +static struct file_system_type devpts_fs_type;
> +
> +struct pts_namespace init_pts_ns = {
> + .kref = {
> + .refcount = ATOMIC_INIT(2),
> + },
> + .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
> + .mnt = NULL,
> +};
>
> static inline struct idr *current_pts_ns_allocated_ptys(void)
> {
> - return &allocated_ptys;
> + return &current->nsproxy->pts_ns->allocated_ptys;
> }
>
> -static struct vfsmount *devpts_mnt;
> static inline struct vfsmount *current_pts_ns_mnt(void)
> {
> - return devpts_mnt;
> + return current->nsproxy->pts_ns->mnt;
> }
>
> static struct {
> @@ -59,6 +66,42 @@ static match_table_t tokens = {
>   {Opt_err, NULL}
> };

>
> +struct pts_namespace *new_pts_ns(void)
> +{
> + struct pts_namespace *ns;
> +
> + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
> + if (!ns)
> + return ERR_PTR(-ENOMEM);
> +
> + ns->mnt = kern_mount_data(&devpts_fs_type, ns);

You create a circular references here - the namespace
holds the vfsmnt, the vfsmnt holds a superblock, a superblock
holds the namespace.

> + if (IS_ERR(ns->mnt)) {
> + kfree(ns);
> + return ERR_PTR(PTR_ERR(ns->mnt));
> + }
> +
> + idr_init(&ns->allocated_ptys);
> + kref_init(&ns->kref);
> +
> + return ns;
> +}
> +
> +void free_pts_ns(struct kref *ns_kref)
> +{
> + struct pts_namespace *ns;
> +
> + ns = container_of(ns_kref, struct pts_namespace, kref);
> + BUG_ON(ns == &init_pts_ns);
> +
> + mntput(ns->mnt);
> + /*
> + * TODO:
> + *   idr_remove_all(&ns->allocated_ptys); introduced in 2.6.23
> + */
> + idr_destroy(&ns->allocated_ptys);
> + kfree(ns);
> +}
> +
> static int devpts_remount(struct super_block *sb, int *flags, char *data)
> {
>   char *p;
> @@ -160,18 +203,27 @@ static int devpts_test_sb(struct super_b
>
> static int devpts_set_sb(struct super_block *sb, void *data)

```
>  {
> - sb->s_fs_info = data;
> + struct pts_namespace *ns = data;
> +
> + sb->s_fs_info = get_pts_ns(ns);
>   return set_anon_super(sb, NULL);
>  }
>
>  static int devpts_get_sb(struct file_system_type *fs_type,
>   int flags, const char *dev_name, void *data, struct vfsmount *mnt)
>  {
> + struct pts_namespace *ns;
>   struct super_block *sb;
>   int err;
>
> + /* hereafter we're very similar to proc_get_sb */
> + if (flags & MS_KERNMOUNT)
> +  ns = data;
> + else
> +  ns = current->nsproxy->pts_ns;
> +
>   /* hereafter we're very simlar to get_sb_nodev */
> - sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
> + sb = sget(fs_type, devpts_test_sb, devpts_set_sb, ns);
>   if (IS_ERR(sb))
>    return PTR_ERR(sb);
>
> @@ -187,16 +239,25 @@ static int devpts_get_sb(struct file_sys
>   }
>
>   sb->s_flags |= MS_ACTIVE;
> - devpts_mnt = mnt;
> + ns->mnt = mnt;
>
>   return simple_set_mnt(mnt, sb);
>  }
>
> +static void devpts_kill_sb(struct super_block *sb)
> +{
> +       struct pts_namespace *ns;
> +
> +       ns = sb->s_fs_info;
> +       kill_anon_super(sb);
> +       put_pts_ns(ns);
> +}
> +
>  static struct file_system_type devpts_fs_type = {
>   .owner  = THIS_MODULE,
```

```
>   .name  = "devpts",
>   .get_sb  = devpts_get_sb,
> - .kill_sb = kill_anon_super,
> + .kill_sb = devpts_kill_sb,
>  };
>
>  /*
> @@ -352,18 +413,19 @@ static int __init init_devpts_fs(void)
>   if (err)
>    return err;
>
> - mnt = kern_mount_data(&devpts_fs_type, NULL);
> + mnt = kern_mount_data(&devpts_fs_type, &init_pts_ns);
>   if (IS_ERR(mnt))
>    err = PTR_ERR(mnt);
>   else
> -  devpts_mnt = mnt;
> +  init_pts_ns.mnt = mnt;
>   return err;
>  }
>
>  static void __exit exit_devpts_fs(void)
>  {
>   unregister_filesystem(&devpts_fs_type);
> - mntput(devpts_mnt);
> + mntput(init_pts_ns.mnt);
> + init_pts_ns.mnt = NULL;
>  }
>
>  module_init(init_devpts_fs)
> Index: linux-2.6.24/include/linux/devpts_fs.h
> ===================================================================
> --- linux-2.6.24.orig/include/linux/devpts_fs.h 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/include/linux/devpts_fs.h 2008-02-05 20:21:08.000000000 -0800
> @@ -14,9 +14,45 @@
>  #define _LINUX_DEVPTS_FS_H
>
>  #include <linux/errno.h>
> +#include <linux/nsproxy.h>
> +#include <linux/kref.h>
> +#include <linux/idr.h>
> +
> +struct pts_namespace {
> + struct kref kref;
> + struct idr allocated_ptys;
> + struct vfsmount *mnt;
> +};
> +
```

```
> +extern struct pts_namespace init_pts_ns;
>
>  #ifdef CONFIG_UNIX98_PTYS
>
> +extern struct pts_namespace *new_pts_ns(void);
> +extern void free_pts_ns(struct kref *kref);
> +
> +static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
> +{
> + if (ns)
> +  kref_get(&ns->kref);
> + return ns;
> +}
> +
> +static inline void put_pts_ns(struct pts_namespace *ns)
> +{
> + if (ns)
> +  kref_put(&ns->kref, free_pts_ns);
> +}
> +
> +static inline struct pts_namespace *copy_pts_ns(unsigned long flags,
> +  struct pts_namespace *old_ns)
> +{
> + if (flags & CLONE_NEWPTS)
> +  return new_pts_ns();
> + else
> +  return get_pts_ns(old_ns);
> +}
> +
>  int devpts_new_index(void);
>  void devpts_kill_index(int idx);
>  int devpts_pty_new(struct tty_struct *tty);      /* mknod in devpts */
> @@ -26,6 +62,22 @@ void devpts_pty_kill(int number);   /* u
>  #else
>
>  /* Dummy stubs in the no-pty case */
> +
> +static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
> +{
> + return &init_pts_ns;
> +}
> +
> +static inline void put_pts_ns(struct pts_namespace *ns) { }
> +
> +static inline struct pts_namespace *copy_pts_ns(unsigned long flags,
> +  struct pts_namespace *old_ns)
> +{
> + if (flags & CLONE_NEWPTS)
```

> + return ERR_PTR(-EINVAL);
> + return old_ns;
> +}
> +
> static inline int devpts_new_index(void) { return -EINVAL; }
> static inline void devpts_kill_index(int idx) { }
> static inline int devpts_pty_new(struct tty_struct *tty) { return -EINVAL; }
> Index: linux-2.6.24/include/linux/init_task.h
> ===
> --- linux-2.6.24.orig/include/linux/init_task.h 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/include/linux/init_task.h 2008-02-05 19:18:00.000000000 -0800
> @@ -77,6 +77,7 @@ extern struct nsproxy init_nsproxy;
>   .mnt_ns = NULL,   \
>   INIT_NET_NS(net_ns)   \
>   INIT_IPC_NS(ipc_ns)   \
> + .pts_ns = &init_pts_ns,   \
>   .user_ns = &init_user_ns,   \
> }
>
> Index: linux-2.6.24/include/linux/nsproxy.h
> ===
> --- linux-2.6.24.orig/include/linux/nsproxy.h 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/include/linux/nsproxy.h 2008-02-05 19:18:00.000000000 -0800
> @@ -8,6 +8,7 @@ struct mnt_namespace;
> struct uts_namespace;
> struct ipc_namespace;
> struct pid_namespace;
> +struct pts_namespace;
>
> /*
>   * A structure to contain pointers to all per-process
> @@ -29,6 +30,7 @@ struct nsproxy {
>   struct pid_namespace *pid_ns;
>   struct user_namespace *user_ns;
>   struct net   *net_ns;
> + struct pts_namespace *pts_ns;
> };
> extern struct nsproxy init_nsproxy;
>
> Index: linux-2.6.24/include/linux/sched.h
> ===
> --- linux-2.6.24.orig/include/linux/sched.h 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/include/linux/sched.h 2008-02-05 19:54:05.000000000 -0800
> @@ -27,6 +27,8 @@
> #define CLONE_NEWUSER 0x10000000 /* New user namespace */
> #define CLONE_NEWPID 0x20000000 /* New pid namespace */
> #define CLONE_NEWNET 0x40000000 /* New network namespace */
> +#define CLONE_NEWPTS   (CLONE_NEWNS|0x80000000) /* Temporary - only for patch

review */
> + /* Badly need to /extend clone() !!! */

:)

> /*
>   * Scheduling policies
> Index: linux-2.6.24/kernel/fork.c
> ===
> --- linux-2.6.24.orig/kernel/fork.c 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/kernel/fork.c 2008-02-05 19:18:00.000000000 -0800
> @@ -1655,7 +1655,7 @@ asmlinkage long sys_unshare(unsigned lon
>   if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
>   CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
>   CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWUSER|
> -   CLONE_NEWNET))
> +   CLONE_NEWNET|CLONE_NEWPTS))
>   goto bad_unshare_out;
>
>   if ((err = unshare_thread(unshare_flags)))
> Index: linux-2.6.24/kernel/nsproxy.c
> ===
> --- linux-2.6.24.orig/kernel/nsproxy.c 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/kernel/nsproxy.c 2008-02-05 19:18:00.000000000 -0800
> @@ -21,6 +21,7 @@
> #include <linux/utsname.h>
> #include <linux/pid_namespace.h>
> #include <net/net_namespace.h>
> +#include <linux/devpts_fs.h>
>
> static struct kmem_cache *nsproxy_cachep;
>
> @@ -92,8 +93,17 @@ static struct nsproxy *create_new_namesp
>   goto out_net;
>   }
>
> + new_nsp->pts_ns = copy_pts_ns(flags, tsk->nsproxy->pts_ns);
> + if (IS_ERR(new_nsp->pts_ns)) {
> + err = PTR_ERR(new_nsp->pts_ns);
> + goto out_pts;
> + }
> +
>   return new_nsp;
>
> +out_pts:
> + if (new_nsp->net_ns)
> + put_net(new_nsp->net_ns);
> out_net:

>   if (new_nsp->user_ns)
>   put_user_ns(new_nsp->user_ns);
> @@ -130,7 +140,8 @@ int copy_namespaces(unsigned long flags,
>   get_nsproxy(old_ns);
>
>   if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
> -   CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET)))
> +   CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET |
> +   CLONE_NEWPTS)))
>   return 0;
>
>   if (!capable(CAP_SYS_ADMIN)) {
> @@ -169,6 +180,8 @@ void free_nsproxy(struct nsproxy *ns)
>   put_pid_ns(ns->pid_ns);
>   if (ns->user_ns)
>   put_user_ns(ns->user_ns);
> + if (ns->pts_ns)
> + put_pts_ns(ns->pts_ns);
>   put_net(ns->net_ns);
>   kmem_cache_free(nsproxy_cachep, ns);
> }
> @@ -183,7 +196,7 @@ int unshare_nsproxy_namespaces(unsigned
>   int err = 0;
>
>   if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
> -   CLONE_NEWUSER | CLONE_NEWNET)))
> +   CLONE_NEWUSER | CLONE_NEWNET | CLONE_NEWPTS)))
>   return 0;
>
>   if (!capable(CAP_SYS_ADMIN))
> _____
> 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 4/4]: Enable cloning PTY namespaces
Posted by serue on Wed, 06 Feb 2008 15:37:50 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> sukadev@us.ibm.com wrote:
> > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

> > Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
> >
> > Enable cloning PTY namespaces.
> >
> > TODO:
> > This version temporarily uses the clone flag '0x80000000' which
> > is unused in mainline atm, but used for CLONE_IO in -mm.
> > While we must extend clone() (urgently) to solve this, it hopefully
> > does not affect review of the rest of this patchset.
> >
> > Changelog:
> > - Version 0: Based on earlier versions from Serge Hallyn and
> >   Matt Helsley.
> >
> > Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> > ---
> > fs/devpts/inode.c | 84 ++++++++++++++++++++++++++++++++++-------
> > include/linux/devpts_fs.h | 52 ++++++++++++++++++++++++++
> > include/linux/init_task.h | 1
> > include/linux/nsproxy.h | 2 +
> > include/linux/sched.h | 2 +
> > kernel/fork.c | 2 -
> > kernel/nsproxy.c | 17 ++++++++-
> > 7 files changed, 146 insertions(+), 14 deletions(-)
> >
> > Index: linux-2.6.24/fs/devpts/inode.c
> > ===
> > --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
> > +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 20:27:41.000000000 -0800
> > @@ -25,18 +25,25 @@
> > #define DEVPTS_SUPER_MAGIC 0x1cd1
> >
> > extern int pty_limit; /* Config limit on Unix98 ptys */
> > -static DEFINE_IDR(allocated_ptys);
> > static DECLARE_MUTEX(allocated_ptys_lock);
> > +static struct file_system_type devpts_fs_type;
> > +
> > +struct pts_namespace init_pts_ns = {
> > + .kref = {
> > + .refcount = ATOMIC_INIT(2),
> > + },
> > + .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
> > + .mnt = NULL,
> > +};
> >
> > static inline struct idr *current_pts_ns_allocated_ptys(void)
> > {
> > - return &allocated_ptys;

> > + return &current->nsproxy->pts_ns->allocated_ptys;
> > }
> >
> > -static struct vfsmount *devpts_mnt;
> > static inline struct vfsmount *current_pts_ns_mnt(void)
> > {
> > - return devpts_mnt;
> > + return current->nsproxy->pts_ns->mnt;
> > }
> >
> > static struct {
> > @@ -59,6 +66,42 @@ static match_table_t tokens = {
> >   {Opt_err, NULL}
> > };
> >
> > +struct pts_namespace *new_pts_ns(void)
> > +{
> > + struct pts_namespace *ns;
> > +
> > + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
> > + if (!ns)
> > + return ERR_PTR(-ENOMEM);
> > +
> > + ns->mnt = kern_mount_data(&devpts_fs_type, ns);
>
> You create a circular references here - the namespace
> holds the vfsmnt, the vfsmnt holds a superblock, a superblock
> holds the namespace.

Hmm, yeah, good point. That was probably in my original version last year, so my fault not Suka's. Suka, would it work to have the sb->s_info point to the namespace but not grab a reference, than have free_pts_ns() null out its sb->s_info, i.e. something like

```
void free_pts_ns(struct kref *ns_kref)
{
   struct pts_namespace *ns;
   struct super_block *sb;

   ns = container_of(ns_kref, struct pts_namespace, kref);
   BUG_ON(ns == &init_pts_ns);
   sb = ns->mnt->mnt_sb;

   mntput(ns->mnt);
   sb->s_info = NULL;

   /*
    * TODO:
```

```
    *      idr_remove_all(&ns->allocated_ptys); introduced in
.6.23
    */
   idr_destroy(&ns->allocated_ptys);
   kfree(ns);
}
```

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by serue on Wed, 06 Feb 2008 15:42:19 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
>
> To support multiple PTY namespaces, we should be allow multiple mounts of
> /dev/pts, once within each PTY namespace.
>
> This patch removes the get_sb_single() in devpts_get_sb() and uses test and
> set sb interfaces to allow remounting /dev/pts. The patch also removes the
> globals, 'devpts_root' and uses current_pts_mnt() to access 'devpts_mnt'
>
> Changelog:
> - Version 0: Based on earlier versions from Serge Hallyn and
>   Matt Helsley.
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

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

Though more of this may be Matt's than mine.

> ---
> fs/devpts/inode.c | 120 ++---------
> 1 file changed, 101 insertions(+), 19 deletions(-)
>
> Index: linux-2.6.24/fs/devpts/inode.c
> ===
> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 17:30:52.000000000 -0800
> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
> @@ -34,7 +34,10 @@ static inline struct idr *current_pts_ns

```
>  }
>
>  static struct vfsmount *devpts_mnt;
> -static struct dentry *devpts_root;
> +static inline struct vfsmount *current_pts_ns_mnt(void)
> +{
> + return devpts_mnt;
> +}
>
>  static struct {
>   int setuid;
> @@ -130,7 +133,7 @@ devpts_fill_super(struct super_block *s,
>   inode->i_fop = &simple_dir_operations;
>   inode->i_nlink = 2;
>
> - devpts_root = s->s_root = d_alloc_root(inode);
> + s->s_root = d_alloc_root(inode);
>   if (s->s_root)
>    return 0;
>
> @@ -140,10 +143,53 @@ fail:
>   return -ENOMEM;
>  }
>
> +/*
> + * We use test and set super-block operations to help determine whether we
> + * need a new super-block for this namespace. get_sb() walks the list of
> + * existing devpts supers, comparing them with the @data ptr. Since we
> + * passed 'current's namespace as the @data pointer we can compare the
> + * namespace pointer in the super-block's 's_fs_info'.  If the test is
> + * TRUE then get_sb() returns a new active reference to the super block.
> + * Otherwise, it helps us build an active reference to a new one.
> + */
> +
> +static int devpts_test_sb(struct super_block *sb, void *data)
> +{
> + return sb->s_fs_info == data;
> +}
> +
> +static int devpts_set_sb(struct super_block *sb, void *data)
> +{
> + sb->s_fs_info = data;
> + return set_anon_super(sb, NULL);
> +}
> +
>  static int devpts_get_sb(struct file_system_type *fs_type,
>   int flags, const char *dev_name, void *data, struct vfsmount *mnt)
>  {
```

> - return get_sb_single(fs_type, flags, data, devpts_fill_super, mnt);
> + struct super_block *sb;
> + int err;
> +
> + /* hereafter we're very simlar to get_sb_nodev */
> + sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
> + if (IS_ERR(sb))
> + return PTR_ERR(sb);
> +
> + if (sb->s_root)
> + return simple_set_mnt(mnt, sb);
> +
> + sb->s_flags = flags;
> + err = devpts_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
> + if (err) {
> + up_write(&sb->s_umount);
> + deactivate_super(sb);
> + return err;
> + }
> +
> + sb->s_flags |= MS_ACTIVE;
> + devpts_mnt = mnt;
> +
> + return simple_set_mnt(mnt, sb);
> }
>
> static struct file_system_type devpts_fs_type = {
> @@ -158,10 +204,9 @@ static struct file_system_type devpts_fs
>   * to the System V naming convention
>   */
>
> -static struct dentry *get_node(int num)
> +static struct dentry *get_node(struct dentry *root, int num)
> {
>   char s[12];
> - struct dentry *root = devpts_root;
>   mutex_lock(&root->d_inode->i_mutex);
>   return lookup_one_len(s, root, sprintf(s, "%d", num));
> }
> @@ -207,12 +252,28 @@ int devpts_pty_new(struct tty_struct *tt
>   struct tty_driver *driver = tty->driver;
>   dev_t device = MKDEV(driver->major, driver->minor_start+number);
>   struct dentry *dentry;
> - struct inode *inode = new_inode(devpts_mnt->mnt_sb);
> + struct dentry *root;
> + struct vfsmount *mnt;
> + struct inode *inode;
> +

```
>
>   /* We're supposed to be given the slave end of a pty */
>   BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
>   BUG_ON(driver->subtype != PTY_TYPE_SLAVE);
>
> + mnt = current_pts_ns_mnt();
> + if (!mnt)
> +  return -ENOSYS;
> + root = mnt->mnt_root;
> +
> + mutex_lock(&root->d_inode->i_mutex);
> + inode = idr_find(current_pts_ns_allocated_ptys(), number);
> + mutex_unlock(&root->d_inode->i_mutex);
> +
> + if (inode && !IS_ERR(inode))
> +  return -EEXIST;
> +
> + inode = new_inode(mnt->mnt_sb);
>   if (!inode)
>    return -ENOMEM;
>
> @@ -222,23 +283,31 @@ int devpts_pty_new(struct tty_struct *tt
>   inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
>   init_special_inode(inode, S_IFCHR|config.mode, device);
>   inode->i_private = tty;
> + idr_replace(current_pts_ns_allocated_ptys(), inode, number);
>
> - dentry = get_node(number);
> + dentry = get_node(root, number);
>   if (!IS_ERR(dentry) && !dentry->d_inode) {
>    d_instantiate(dentry, inode);
> -  fsnotify_create(devpts_root->d_inode, dentry);
> +  fsnotify_create(root->d_inode, dentry);
>   }
>
> - mutex_unlock(&devpts_root->d_inode->i_mutex);
> + mutex_unlock(&root->d_inode->i_mutex);
>
>   return 0;
>  }
>
>  struct tty_struct *devpts_get_tty(int number)
>  {
> - struct dentry *dentry = get_node(number);
> + struct vfsmount *mnt;
> + struct dentry *dentry;
>   struct tty_struct *tty;
>
```

```
> + mnt = current_pts_ns_mnt();
> + if (!mnt)
> +  return NULL;
> +
> + dentry = get_node(mnt->mnt_root, number);
> +
>   tty = NULL;
>   if (!IS_ERR(dentry)) {
>    if (dentry->d_inode)
> @@ -246,14 +315,21 @@ struct tty_struct *devpts_get_tty(int nu
>    dput(dentry);
>   }
>
> - mutex_unlock(&devpts_root->d_inode->i_mutex);
> + mutex_unlock(&mnt->mnt_root->d_inode->i_mutex);
>
>   return tty;
>  }
>
>  void devpts_pty_kill(int number)
>  {
> - struct dentry *dentry = get_node(number);
> + struct dentry *dentry;
> + struct dentry *root;
> + struct vfsmount *mnt;
> +
> + mnt = current_pts_ns_mnt();
> + root = mnt->mnt_root;
> +
> + dentry = get_node(root, number);
>
>   if (!IS_ERR(dentry)) {
>    struct inode *inode = dentry->d_inode;
> @@ -264,17 +340,23 @@ void devpts_pty_kill(int number)
>    }
>    dput(dentry);
>   }
> - mutex_unlock(&devpts_root->d_inode->i_mutex);
> + mutex_unlock(&root->d_inode->i_mutex);
>  }
>
>  static int __init init_devpts_fs(void)
>  {
> - int err = register_filesystem(&devpts_fs_type);
> - if (!err) {
> -  devpts_mnt = kern_mount(&devpts_fs_type);
> -  if (IS_ERR(devpts_mnt))
> -   err = PTR_ERR(devpts_mnt);
```

> - }
> + struct vfsmount *mnt;
> + int err;
> +
> + err = register_filesystem(&devpts_fs_type);
> + if (err)
> + return err;
> +
> + mnt = kern_mount_data(&devpts_fs_type, NULL);
> + if (IS_ERR(mnt))
> + err = PTR_ERR(mnt);
> + else
> + devpts_mnt = mnt;
>   return err;
> }
>
> _____
> 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 4/4]: Enable cloning PTY namespaces
Posted by Pavel Emelianov on Wed, 06 Feb 2008 15:44:58 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> sukadev@us.ibm.com wrote:
>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>> Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
>>>
>>> Enable cloning PTY namespaces.
>>>
>>> TODO:
>>> This version temporarily uses the clone flag '0x80000000' which
>>> is unused in mainline atm, but used for CLONE_IO in -mm.
>>> While we must extend clone() (urgently) to solve this, it hopefully
>>> does not affect review of the rest of this patchset.
>>>
>>> Changelog:
>>> - Version 0: Based on earlier versions from Serge Hallyn and
>>>   Matt Helsley.
>>>

```
>>> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>> ---
>>>  fs/devpts/inode.c         |   84 ++++++++++++++++++++++++++++++++++++++-------
>>>  include/linux/devpts_fs.h |   52 ++++++++++++++++++++++++++++
>>>  include/linux/init_task.h |    1
>>>  include/linux/nsproxy.h   |    2 +
>>>  include/linux/sched.h     |    2 +
>>>  kernel/fork.c             |    2 -
>>>  kernel/nsproxy.c          |   17 ++++++++-
>>>  7 files changed, 146 insertions(+), 14 deletions(-)
>>>
>>> Index: linux-2.6.24/fs/devpts/inode.c
>>> ===================================================================
>>> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
>>> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 20:27:41.000000000 -0800
>>> @@ -25,18 +25,25 @@
>>>  #define DEVPTS_SUPER_MAGIC 0x1cd1
>>>
>>>  extern int pty_limit;  /* Config limit on Unix98 ptys */
>>> -static DEFINE_IDR(allocated_ptys);
>>>  static DECLARE_MUTEX(allocated_ptys_lock);
>>> +static struct file_system_type devpts_fs_type;
>>> +
>>> +struct pts_namespace init_pts_ns = {
>>> + .kref = {
>>> +  .refcount = ATOMIC_INIT(2),
>>> + },
>>> + .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
>>> + .mnt = NULL,
>>> +};
>>>
>>>  static inline struct idr *current_pts_ns_allocated_ptys(void)
>>>  {
>>> - return &allocated_ptys;
>>> + return &current->nsproxy->pts_ns->allocated_ptys;
>>>  }
>>>
>>> -static struct vfsmount *devpts_mnt;
>>>  static inline struct vfsmount *current_pts_ns_mnt(void)
>>>  {
>>> - return devpts_mnt;
>>> + return current->nsproxy->pts_ns->mnt;
>>>  }
>>>
>>>  static struct {
>>> @@ -59,6 +66,42 @@ static match_table_t tokens = {
>>>   {Opt_err, NULL}
>>>  };
```

>>>
>>> +struct pts_namespace *new_pts_ns(void)
>>> +{
>>> + struct pts_namespace *ns;
>>> +
>>> + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
>>> + if (!ns)
>>> + return ERR_PTR(-ENOMEM);
>>> +
>>> + ns->mnt = kern_mount_data(&devpts_fs_type, ns);
>> You create a circular references here - the namespace
>> holds the vfsmnt, the vfsmnt holds a superblock, a superblock
>> holds the namespace.
>
> Hmm, yeah, good point. That was probably in my original version last
> year, so my fault not Suka's. Suka, would it work to have the
> sb->s_info point to the namespace but not grab a reference, than have

If you don't then you may be in situation, when this devpts
is mounted from userspace and in case the namespace is dead
superblock will point to garbage... Superblock MUST hold the
namespace :)

> free_pts_ns() null out its sb->s_info, i.e. something like
>
> void free_pts_ns(struct kref *ns_kref)
> {
>   struct pts_namespace *ns;
>   struct super_block *sb;
>
>   ns = container_of(ns_kref, struct pts_namespace, kref);
>   BUG_ON(ns == &init_pts_ns);
>   sb = ns->mnt->mnt_sb;
>
>   mntput(ns->mnt);
>   sb->s_info = NULL;
>
>   /*
>   * TODO:
>   *   idr_remove_all(&ns->allocated_ptys); introduced in
> .6.23
>   */
>   idr_destroy(&ns->allocated_ptys);
>   kfree(ns);
> }
>
>

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Pavel Emelianov on Wed, 06 Feb 2008 15:57:08 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
>
> To support multiple PTY namespaces, we should be allow multiple mounts of
> /dev/pts, once within each PTY namespace.
>
> This patch removes the get_sb_single() in devpts_get_sb() and uses test and
> set sb interfaces to allow remounting /dev/pts. The patch also removes the
> globals, 'devpts_root' and uses current_pts_mnt() to access 'devpts_mnt'
>
> Changelog:
> - Version 0: Based on earlier versions from Serge Hallyn and
> Matt Helsley.
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> fs/devpts/inode.c | 120 ++---------
> 1 file changed, 101 insertions(+), 19 deletions(-)
>
> Index: linux-2.6.24/fs/devpts/inode.c
> ===
> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 17:30:52.000000000 -0800
> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
> @@ -34,7 +34,10 @@ static inline struct idr *current_pts_ns
> }
>
> static struct vfsmount *devpts_mnt;
> -static struct dentry *devpts_root;
> +static inline struct vfsmount *current_pts_ns_mnt(void)
> +{
> + return devpts_mnt;
> +}
>
> static struct {
> int setuid;
> @@ -130,7 +133,7 @@ devpts_fill_super(struct super_block *s,
> inode->i_fop = &simple_dir_operations;

>   inode->i_nlink = 2;
>
> - devpts_root = s->s_root = d_alloc_root(inode);
> + s->s_root = d_alloc_root(inode);
>   if (s->s_root)
>   return 0;
>
> @@ -140,10 +143,53 @@ fail:
>   return -ENOMEM;
> }
>
> +/*
> + * We use test and set super-block operations to help determine whether we
> + * need a new super-block for this namespace. get_sb() walks the list of
> + * existing devpts supers, comparing them with the @data ptr. Since we
> + * passed 'current's namespace as the @data pointer we can compare the
> + * namespace pointer in the super-block's 's_fs_info'. If the test is
> + * TRUE then get_sb() returns a new active reference to the super block.
> + * Otherwise, it helps us build an active reference to a new one.
> + */
> +
> +static int devpts_test_sb(struct super_block *sb, void *data)
> +{
> + return sb->s_fs_info == data;
> +}
> +
> +static int devpts_set_sb(struct super_block *sb, void *data)
> +{
> + sb->s_fs_info = data;
> + return set_anon_super(sb, NULL);
> +}
> +
> static int devpts_get_sb(struct file_system_type *fs_type,
>   int flags, const char *dev_name, void *data, struct vfsmount *mnt)
> {
> - return get_sb_single(fs_type, flags, data, devpts_fill_super, mnt);
> + struct super_block *sb;
> + int err;
> +
> + /* hereafter we're very simlar to get_sb_nodev */
> + sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
> + if (IS_ERR(sb))
> + return PTR_ERR(sb);
> +
> + if (sb->s_root)
> + return simple_set_mnt(mnt, sb);
> +
> + sb->s_flags = flags;

> + err = devpts_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
> + if (err) {
> + up_write(&sb->s_umount);
> + deactivate_super(sb);
> + return err;
> + }
> +

That stuff becomes very very similar to that in proc :)
Makes sense to consolidate. Maybe...

> + sb->s_flags |= MS_ACTIVE;
> + devpts_mnt = mnt;
> +
> + return simple_set_mnt(mnt, sb);
> }
>
> static struct file_system_type devpts_fs_type = {
> @@ -158,10 +204,9 @@ static struct file_system_type devpts_fs
>   * to the System V naming convention
>   */
>
> -static struct dentry *get_node(int num)
> +static struct dentry *get_node(struct dentry *root, int num)
> {
>   char s[12];
> - struct dentry *root = devpts_root;
>   mutex_lock(&root->d_inode->i_mutex);
>   return lookup_one_len(s, root, sprintf(s, "%d", num));
> }
> @@ -207,12 +252,28 @@ int devpts_pty_new(struct tty_struct *tt
>   struct tty_driver *driver = tty->driver;
>   dev_t device = MKDEV(driver->major, driver->minor_start+number);
>   struct dentry *dentry;
> - struct inode *inode = new_inode(devpts_mnt->mnt_sb);
> + struct dentry *root;
> + struct vfsmount *mnt;
> + struct inode *inode;
> +
>
>   /* We're supposed to be given the slave end of a pty */
>   BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
>   BUG_ON(driver->subtype != PTY_TYPE_SLAVE);
>
> + mnt = current_pts_ns_mnt();
> + if (!mnt)
> + return -ENOSYS;
> + root = mnt->mnt_root;

```
> +
> + mutex_lock(&root->d_inode->i_mutex);
> + inode = idr_find(current_pts_ns_allocated_ptys(), number);
> + mutex_unlock(&root->d_inode->i_mutex);
> +
> + if (inode && !IS_ERR(inode))
> +  return -EEXIST;
> +
> + inode = new_inode(mnt->mnt_sb);
>   if (!inode)
>    return -ENOMEM;
>
> @@ -222,23 +283,31 @@ int devpts_pty_new(struct tty_struct *tt
>   inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
>   init_special_inode(inode, S_IFCHR|config.mode, device);
>   inode->i_private = tty;
> + idr_replace(current_pts_ns_allocated_ptys(), inode, number);
>
> - dentry = get_node(number);
> + dentry = get_node(root, number);
>   if (!IS_ERR(dentry) && !dentry->d_inode) {
>    d_instantiate(dentry, inode);
> -  fsnotify_create(devpts_root->d_inode, dentry);
> +  fsnotify_create(root->d_inode, dentry);
>   }
>
> - mutex_unlock(&devpts_root->d_inode->i_mutex);
> + mutex_unlock(&root->d_inode->i_mutex);
>
>   return 0;
>  }
>
>  struct tty_struct *devpts_get_tty(int number)
>  {
> - struct dentry *dentry = get_node(number);
> + struct vfsmount *mnt;
> + struct dentry *dentry;
>   struct tty_struct *tty;
>
> + mnt = current_pts_ns_mnt();
> + if (!mnt)
> +  return NULL;
> +
> + dentry = get_node(mnt->mnt_root, number);
> +
>   tty = NULL;
>   if (!IS_ERR(dentry)) {
>    if (dentry->d_inode)
```

```
> @@ -246,14 +315,21 @@ struct tty_struct *devpts_get_tty(int nu
>    dput(dentry);
>   }
>
> - mutex_unlock(&devpts_root->d_inode->i_mutex);
> + mutex_unlock(&mnt->mnt_root->d_inode->i_mutex);
>
>   return tty;
>  }
>
>  void devpts_pty_kill(int number)
>  {
> - struct dentry *dentry = get_node(number);
> + struct dentry *dentry;
> + struct dentry *root;
> + struct vfsmount *mnt;
> +
> + mnt = current_pts_ns_mnt();
> + root = mnt->mnt_root;
> +
> + dentry = get_node(root, number);
>
>   if (!IS_ERR(dentry)) {
>    struct inode *inode = dentry->d_inode;
> @@ -264,17 +340,23 @@ void devpts_pty_kill(int number)
>    }
>    dput(dentry);
>   }
> - mutex_unlock(&devpts_root->d_inode->i_mutex);
> + mutex_unlock(&root->d_inode->i_mutex);
>  }
>
>  static int __init init_devpts_fs(void)
>  {
> - int err = register_filesystem(&devpts_fs_type);
> - if (!err) {
> -  devpts_mnt = kern_mount(&devpts_fs_type);
> -  if (IS_ERR(devpts_mnt))
> -   err = PTR_ERR(devpts_mnt);
> - }
> + struct vfsmount *mnt;
> + int err;
> +
> + err = register_filesystem(&devpts_fs_type);
> + if (err)
> +  return err;
> +
> + mnt = kern_mount_data(&devpts_fs_type, NULL);
```

> + if (IS_ERR(mnt))
> + err = PTR_ERR(mnt);
> + else
> + devpts_mnt = mnt;
>   return err;
> }
>
> _____
> 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 4/4]: Enable cloning PTY namespaces
Posted by serue on Wed, 06 Feb 2008 16:03:28 GMT
View Forum Message <> Reply to Message

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
>
> Enable cloning PTY namespaces.
>
> TODO:
> This version temporarily uses the clone flag '0x80000000' which
> is unused in mainline atm, but used for CLONE_IO in -mm.
> While we must extend clone() (urgently) to solve this, it hopefully
> does not affect review of the rest of this patchset.
>
> Changelog:
> - Version 0: Based on earlier versions from Serge Hallyn and
>   Matt Helsley.
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>

Thanks for carrying this forward, Suka, and Matt.

Of course it still needs at least Pavel's concern addressed, but

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

to start the SOB chain.

thanks,
-serge

> ---
> fs/devpts/inode.c | 84 +++++++++++++++++++++++++++++++++++++++-------
> include/linux/devpts_fs.h | 52 ++++++++++++++++++++++++++++
> include/linux/init_task.h | 1
> include/linux/nsproxy.h | 2 +
> include/linux/sched.h | 2 +
> kernel/fork.c | 2 -
> kernel/nsproxy.c | 17 ++++++++-
> 7 files changed, 146 insertions(+), 14 deletions(-)
>
> Index: linux-2.6.24/fs/devpts/inode.c
> ===
> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 20:27:41.000000000 -0800
> @@ -25,18 +25,25 @@
> #define DEVPTS_SUPER_MAGIC 0x1cd1
>
> extern int pty_limit; /* Config limit on Unix98 ptys */
> -static DEFINE_IDR(allocated_ptys);
> static DECLARE_MUTEX(allocated_ptys_lock);
> +static struct file_system_type devpts_fs_type;
> +
> +struct pts_namespace init_pts_ns = {
> + .kref = {
> + .refcount = ATOMIC_INIT(2),
> + },
> + .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
> + .mnt = NULL,
> +};
>
> static inline struct idr *current_pts_ns_allocated_ptys(void)
> {
> - return &allocated_ptys;
> + return &current->nsproxy->pts_ns->allocated_ptys;
> }
>
> -static struct vfsmount *devpts_mnt;
> static inline struct vfsmount *current_pts_ns_mnt(void)
> {
> - return devpts_mnt;
> + return current->nsproxy->pts_ns->mnt;
> }
>
> static struct {
> @@ -59,6 +66,42 @@ static match_table_t tokens = {

> {Opt_err, NULL}
> };
>
> +struct pts_namespace *new_pts_ns(void)
> +{
> + struct pts_namespace *ns;
> +
> + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
> + if (!ns)
> + return ERR_PTR(-ENOMEM);
> +
> + ns->mnt = kern_mount_data(&devpts_fs_type, ns);
> + if (IS_ERR(ns->mnt)) {
> + kfree(ns);
> + return ERR_PTR(PTR_ERR(ns->mnt));
> + }
> +
> + idr_init(&ns->allocated_ptys);
> + kref_init(&ns->kref);
> +
> + return ns;
> +}
> +
> +void free_pts_ns(struct kref *ns_kref)
> +{
> + struct pts_namespace *ns;
> +
> + ns = container_of(ns_kref, struct pts_namespace, kref);
> + BUG_ON(ns == &init_pts_ns);
> +
> + mntput(ns->mnt);
> + /*
> + * TODO:
> + *   idr_remove_all(&ns->allocated_ptys); introduced in 2.6.23
> + */
> + idr_destroy(&ns->allocated_ptys);
> + kfree(ns);
> +}
> +
> static int devpts_remount(struct super_block *sb, int *flags, char *data)
> {
>   char *p;
> @@ -160,18 +203,27 @@ static int devpts_test_sb(struct super_b
>
> static int devpts_set_sb(struct super_block *sb, void *data)
> {
> - sb->s_fs_info = data;
> + struct pts_namespace *ns = data;

> +
> + sb->s_fs_info = get_pts_ns(ns);
> return set_anon_super(sb, NULL);
> }
>
> static int devpts_get_sb(struct file_system_type *fs_type,
> int flags, const char *dev_name, void *data, struct vfsmount *mnt)
> {
> + struct pts_namespace *ns;
> struct super_block *sb;
> int err;
>
> + /* hereafter we're very similar to proc_get_sb */
> + if (flags & MS_KERNMOUNT)
> + ns = data;
> + else
> + ns = current->nsproxy->pts_ns;
> +
> /* hereafter we're very simlar to get_sb_nodev */
> - sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
> + sb = sget(fs_type, devpts_test_sb, devpts_set_sb, ns);
> if (IS_ERR(sb))
> return PTR_ERR(sb);
>
> @@ -187,16 +239,25 @@ static int devpts_get_sb(struct file_sys
> }
>
> sb->s_flags |= MS_ACTIVE;
> - devpts_mnt = mnt;
> + ns->mnt = mnt;
>
> return simple_set_mnt(mnt, sb);
> }
>
> +static void devpts_kill_sb(struct super_block *sb)
> +{
> + struct pts_namespace *ns;
> +
> + ns = sb->s_fs_info;
> + kill_anon_super(sb);
> + put_pts_ns(ns);
> +}
> +
> static struct file_system_type devpts_fs_type = {
> .owner = THIS_MODULE,
> .name = "devpts",
> .get_sb = devpts_get_sb,
> - .kill_sb = kill_anon_super,

```
> + .kill_sb = devpts_kill_sb,
>  };
>
>  /*
> @@ -352,18 +413,19 @@ static int __init init_devpts_fs(void)
>   if (err)
>    return err;
>
> - mnt = kern_mount_data(&devpts_fs_type, NULL);
> + mnt = kern_mount_data(&devpts_fs_type, &init_pts_ns);
>   if (IS_ERR(mnt))
>    err = PTR_ERR(mnt);
>   else
> -  devpts_mnt = mnt;
> +  init_pts_ns.mnt = mnt;
>   return err;
>  }
>
>  static void __exit exit_devpts_fs(void)
>  {
>   unregister_filesystem(&devpts_fs_type);
> - mntput(devpts_mnt);
> + mntput(init_pts_ns.mnt);
> + init_pts_ns.mnt = NULL;
>  }
>
>  module_init(init_devpts_fs)
> Index: linux-2.6.24/include/linux/devpts_fs.h
> ===================================================================
> --- linux-2.6.24.orig/include/linux/devpts_fs.h 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/include/linux/devpts_fs.h 2008-02-05 20:21:08.000000000 -0800
> @@ -14,9 +14,45 @@
>  #define _LINUX_DEVPTS_FS_H
>
>  #include <linux/errno.h>
> +#include <linux/nsproxy.h>
> +#include <linux/kref.h>
> +#include <linux/idr.h>
> +
> +struct pts_namespace {
> + struct kref kref;
> + struct idr allocated_ptys;
> + struct vfsmount *mnt;
> +};
> +
> +extern struct pts_namespace init_pts_ns;
>
>  #ifdef CONFIG_UNIX98_PTYS
```

>
> +extern struct pts_namespace *new_pts_ns(void);
> +extern void free_pts_ns(struct kref *kref);
> +
> +static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
> +{
> + if (ns)
> + kref_get(&ns->kref);
> + return ns;
> +}
> +
> +static inline void put_pts_ns(struct pts_namespace *ns)
> +{
> + if (ns)
> + kref_put(&ns->kref, free_pts_ns);
> +}
> +
> +static inline struct pts_namespace *copy_pts_ns(unsigned long flags,
> + struct pts_namespace *old_ns)
> +{
> + if (flags & CLONE_NEWPTS)
> + return new_pts_ns();
> + else
> + return get_pts_ns(old_ns);
> +}
> +
> int devpts_new_index(void);
> void devpts_kill_index(int idx);
> int devpts_pty_new(struct tty_struct *tty);   /* mknod in devpts */
> @@ -26,6 +62,22 @@ void devpts_pty_kill(int number);   /* u
> #else
>
> /* Dummy stubs in the no-pty case */
> +
> +static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
> +{
> + return &init_pts_ns;
> +}
> +
> +static inline void put_pts_ns(struct pts_namespace *ns) { }
> +
> +static inline struct pts_namespace *copy_pts_ns(unsigned long flags,
> + struct pts_namespace *old_ns)
> +{
> + if (flags & CLONE_NEWPTS)
> + return ERR_PTR(-EINVAL);
> + return old_ns;
> +}

> +
> static inline int devpts_new_index(void) { return -EINVAL; }
> static inline void devpts_kill_index(int idx) { }
> static inline int devpts_pty_new(struct tty_struct *tty) { return -EINVAL; }
> Index: linux-2.6.24/include/linux/init_task.h
> ===
> --- linux-2.6.24.orig/include/linux/init_task.h 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/include/linux/init_task.h 2008-02-05 19:18:00.000000000 -0800
> @@ -77,6 +77,7 @@ extern struct nsproxy init_nsproxy;
>   .mnt_ns = NULL,   \
>   INIT_NET_NS(net_ns)   \
>   INIT_IPC_NS(ipc_ns)   \
> + .pts_ns = &init_pts_ns,   \
>   .user_ns = &init_user_ns,   \
> }
>
> Index: linux-2.6.24/include/linux/nsproxy.h
> ===
> --- linux-2.6.24.orig/include/linux/nsproxy.h 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/include/linux/nsproxy.h 2008-02-05 19:18:00.000000000 -0800
> @@ -8,6 +8,7 @@ struct mnt_namespace;
> struct uts_namespace;
> struct ipc_namespace;
> struct pid_namespace;
> +struct pts_namespace;
>
> /*
>   * A structure to contain pointers to all per-process
> @@ -29,6 +30,7 @@ struct nsproxy {
>   struct pid_namespace *pid_ns;
>   struct user_namespace *user_ns;
>   struct net   *net_ns;
> + struct pts_namespace *pts_ns;
> };
> extern struct nsproxy init_nsproxy;
>
> Index: linux-2.6.24/include/linux/sched.h
> ===
> --- linux-2.6.24.orig/include/linux/sched.h 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/include/linux/sched.h 2008-02-05 19:54:05.000000000 -0800
> @@ -27,6 +27,8 @@
> #define CLONE_NEWUSER 0x10000000 /* New user namespace */
> #define CLONE_NEWPID 0x20000000 /* New pid namespace */
> #define CLONE_NEWNET 0x40000000 /* New network namespace */
> +#define CLONE_NEWPTS   (CLONE_NEWNS|0x80000000) /* Temporary - only for patch review */
> +   /* Badly need to /extend clone() !!! */
>

```
>  /*
>   * Scheduling policies
> Index: linux-2.6.24/kernel/fork.c
> ===================================================================
> --- linux-2.6.24.orig/kernel/fork.c 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/kernel/fork.c 2008-02-05 19:18:00.000000000 -0800
> @@ -1655,7 +1655,7 @@ asmlinkage long sys_unshare(unsigned lon
>   if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
>       CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
>       CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWUSER|
> -     CLONE_NEWNET))
> +     CLONE_NEWNET|CLONE_NEWPTS))
>     goto bad_unshare_out;
>
>   if ((err = unshare_thread(unshare_flags)))
> Index: linux-2.6.24/kernel/nsproxy.c
> ===================================================================
> --- linux-2.6.24.orig/kernel/nsproxy.c 2008-02-05 19:16:39.000000000 -0800
> +++ linux-2.6.24/kernel/nsproxy.c 2008-02-05 19:18:00.000000000 -0800
> @@ -21,6 +21,7 @@
>  #include <linux/utsname.h>
>  #include <linux/pid_namespace.h>
>  #include <net/net_namespace.h>
> +#include <linux/devpts_fs.h>
>
>  static struct kmem_cache *nsproxy_cachep;
>
> @@ -92,8 +93,17 @@ static struct nsproxy *create_new_namesp
>     goto out_net;
>   }
>
> + new_nsp->pts_ns = copy_pts_ns(flags, tsk->nsproxy->pts_ns);
> + if (IS_ERR(new_nsp->pts_ns)) {
> +  err = PTR_ERR(new_nsp->pts_ns);
> +  goto out_pts;
> + }
> +
>   return new_nsp;
>
> +out_pts:
> + if (new_nsp->net_ns)
> +  put_net(new_nsp->net_ns);
>  out_net:
>   if (new_nsp->user_ns)
>     put_user_ns(new_nsp->user_ns);
> @@ -130,7 +140,8 @@ int copy_namespaces(unsigned long flags,
>   get_nsproxy(old_ns);
>
```

>   if (!(flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
> -   CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET)))
> +   CLONE_NEWUSER | CLONE_NEWPID | CLONE_NEWNET |
> +   CLONE_NEWPTS)))
>   return 0;
>
>   if (!capable(CAP_SYS_ADMIN)) {
> @@ -169,6 +180,8 @@ void free_nsproxy(struct nsproxy *ns)
>   put_pid_ns(ns->pid_ns);
>   if (ns->user_ns)
>   put_user_ns(ns->user_ns);
> + if (ns->pts_ns)
> + put_pts_ns(ns->pts_ns);
>   put_net(ns->net_ns);
>   kmem_cache_free(nsproxy_cachep, ns);
> }
> @@ -183,7 +196,7 @@ int unshare_nsproxy_namespaces(unsigned
>   int err = 0;
>
>   if (!(unshare_flags & (CLONE_NEWNS | CLONE_NEWUTS | CLONE_NEWIPC |
> -   CLONE_NEWUSER | CLONE_NEWNET)))
> +   CLONE_NEWUSER | CLONE_NEWNET | CLONE_NEWPTS)))
>   return 0;
>
>   if (!capable(CAP_SYS_ADMIN))
> _____
> 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 3/4]: Enable multiple mounts of /dev/pts
Posted by serue on Wed, 06 Feb 2008 16:16:08 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> sukadev@us.ibm.com wrote:
> > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> > Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
> >
> > To support multiple PTY namespaces, we should be allow multiple mounts of
> > /dev/pts, once within each PTY namespace.
> >
> > This patch removes the get_sb_single() in devpts_get_sb() and uses test and

> > set sb interfaces to allow remounting /dev/pts. The patch also removes the
> > globals, 'devpts_root' and uses current_pts_mnt() to access 'devpts_mnt'
> >
> > Changelog:
> > - Version 0: Based on earlier versions from Serge Hallyn and
> >   Matt Helsley.
> >
> > Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> > ---
> > fs/devpts/inode.c | 120 ++---------
> > 1 file changed, 101 insertions(+), 19 deletions(-)
> >
> > Index: linux-2.6.24/fs/devpts/inode.c
> > ===
> > --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 17:30:52.000000000 -0800
> > +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
> > @@ -34,7 +34,10 @@ static inline struct idr *current_pts_ns
> > }
> >
> > static struct vfsmount *devpts_mnt;
> > -static struct dentry *devpts_root;
> > +static inline struct vfsmount *current_pts_ns_mnt(void)
> > +{
> > + return devpts_mnt;
> > +}
> >
> > static struct {
> >   int setuid;
> > @@ -130,7 +133,7 @@ devpts_fill_super(struct super_block *s,
> >   inode->i_fop = &simple_dir_operations;
> >   inode->i_nlink = 2;
> >
> > - devpts_root = s->s_root = d_alloc_root(inode);
> > + s->s_root = d_alloc_root(inode);
> >   if (s->s_root)
> >   return 0;
> >
> > @@ -140,10 +143,53 @@ fail:
> >   return -ENOMEM;
> > }
> >
> > +/*
> > + * We use test and set super-block operations to help determine whether we
> > + * need a new super-block for this namespace. get_sb() walks the list of
> > + * existing devpts supers, comparing them with the @data ptr. Since we
> > + * passed 'current's namespace as the @data pointer we can compare the
> > + * namespace pointer in the super-block's 's_fs_info'. If the test is
> > + * TRUE then get_sb() returns a new active reference to the super block.

```
> > + * Otherwise, it helps us build an active reference to a new one.
> > + */
> > +
> > +static int devpts_test_sb(struct super_block *sb, void *data)
> > +{
> > + return sb->s_fs_info == data;
> > +}
> > +
> > +static int devpts_set_sb(struct super_block *sb, void *data)
> > +{
> > + sb->s_fs_info = data;
> > + return set_anon_super(sb, NULL);
> > +}
> > +
> >  static int devpts_get_sb(struct file_system_type *fs_type,
> >   int flags, const char *dev_name, void *data, struct vfsmount *mnt)
> >  {
> > - return get_sb_single(fs_type, flags, data, devpts_fill_super, mnt);
> > + struct super_block *sb;
> > + int err;
> > +
> > + /* hereafter we're very simlar to get_sb_nodev */
> > + sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
> > + if (IS_ERR(sb))
> > +  return PTR_ERR(sb);
> > +
> > + if (sb->s_root)
> > +  return simple_set_mnt(mnt, sb);
> > +
> > + sb->s_flags = flags;
> > + err = devpts_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
> > + if (err) {
> > +  up_write(&sb->s_umount);
> > +  deactivate_super(sb);
> > +  return err;
> > + }
> > +
>
> That stuff becomes very very similar to that in proc :)
> Makes sense to consolidate. Maybe...
```

Yeah, and the mqns that Cedric sent too. I think Cedric said he'd started an a patch implementing a helper. Cedric?

Pavel, not long ago you said you were starting to look at tty and pty stuff - did you have any different ideas on devpts virtualization, or are you ok with this minus your comments thus far?

```
>
> > + sb->s_flags |= MS_ACTIVE;
> > + devpts_mnt = mnt;
> > +
> > + return simple_set_mnt(mnt, sb);
> >  }
> >
> >  static struct file_system_type devpts_fs_type = {
> > @@ -158,10 +204,9 @@ static struct file_system_type devpts_fs
> >   * to the System V naming convention
> >   */
> >
> > -static struct dentry *get_node(int num)
> > +static struct dentry *get_node(struct dentry *root, int num)
> >  {
> >   char s[12];
> > - struct dentry *root = devpts_root;
> >   mutex_lock(&root->d_inode->i_mutex);
> >   return lookup_one_len(s, root, sprintf(s, "%d", num));
> >  }
> > @@ -207,12 +252,28 @@ int devpts_pty_new(struct tty_struct *tt
> >   struct tty_driver *driver = tty->driver;
> >   dev_t device = MKDEV(driver->major, driver->minor_start+number);
> >   struct dentry *dentry;
> > - struct inode *inode = new_inode(devpts_mnt->mnt_sb);
> > + struct dentry *root;
> > + struct vfsmount *mnt;
> > + struct inode *inode;
> > +
> >
> >   /* We're supposed to be given the slave end of a pty */
> >   BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
> >   BUG_ON(driver->subtype != PTY_TYPE_SLAVE);
> >
> > + mnt = current_pts_ns_mnt();
> > + if (!mnt)
> > +  return -ENOSYS;
> > + root = mnt->mnt_root;
> > +
> > + mutex_lock(&root->d_inode->i_mutex);
> > + inode = idr_find(current_pts_ns_allocated_ptys(), number);
> > + mutex_unlock(&root->d_inode->i_mutex);
> > +
> > + if (inode && !IS_ERR(inode))
> > +  return -EEXIST;
> > +
> > + inode = new_inode(mnt->mnt_sb);
> >   if (!inode)
```

```
> >    return -ENOMEM;
> >
> > @@ -222,23 +283,31 @@ int devpts_pty_new(struct tty_struct *tt
> >   inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
> >   init_special_inode(inode, S_IFCHR|config.mode, device);
> >   inode->i_private = tty;
> > + idr_replace(current_pts_ns_allocated_ptys(), inode, number);
> >
> > - dentry = get_node(number);
> > + dentry = get_node(root, number);
> >   if (!IS_ERR(dentry) && !dentry->d_inode) {
> >    d_instantiate(dentry, inode);
> > -  fsnotify_create(devpts_root->d_inode, dentry);
> > +  fsnotify_create(root->d_inode, dentry);
> >   }
> >
> > - mutex_unlock(&devpts_root->d_inode->i_mutex);
> > + mutex_unlock(&root->d_inode->i_mutex);
> >
> >   return 0;
> >  }
> >
> >  struct tty_struct *devpts_get_tty(int number)
> >  {
> > - struct dentry *dentry = get_node(number);
> > + struct vfsmount *mnt;
> > + struct dentry *dentry;
> >   struct tty_struct *tty;
> >
> > + mnt = current_pts_ns_mnt();
> > + if (!mnt)
> > +  return NULL;
> > +
> > + dentry = get_node(mnt->mnt_root, number);
> > +
> >   tty = NULL;
> >   if (!IS_ERR(dentry)) {
> >    if (dentry->d_inode)
> > @@ -246,14 +315,21 @@ struct tty_struct *devpts_get_tty(int nu
> >    dput(dentry);
> >   }
> >
> > - mutex_unlock(&devpts_root->d_inode->i_mutex);
> > + mutex_unlock(&mnt->mnt_root->d_inode->i_mutex);
> >
> >   return tty;
> >  }
> >
```

> > void devpts_pty_kill(int number)
> > {
> > - struct dentry *dentry = get_node(number);
> > + struct dentry *dentry;
> > + struct dentry *root;
> > + struct vfsmount *mnt;
> > +
> > + mnt = current_pts_ns_mnt();
> > + root = mnt->mnt_root;
> > +
> > + dentry = get_node(root, number);
> >
> >   if (!IS_ERR(dentry)) {
> >   struct inode *inode = dentry->d_inode;
> > @@ -264,17 +340,23 @@ void devpts_pty_kill(int number)
> >   }
> >   dput(dentry);
> >   }
> > - mutex_unlock(&devpts_root->d_inode->i_mutex);
> > + mutex_unlock(&root->d_inode->i_mutex);
> > }
> >
> > static int __init init_devpts_fs(void)
> > {
> > - int err = register_filesystem(&devpts_fs_type);
> > - if (!err) {
> > - devpts_mnt = kern_mount(&devpts_fs_type);
> > - if (IS_ERR(devpts_mnt))
> > -   err = PTR_ERR(devpts_mnt);
> > - }
> > + struct vfsmount *mnt;
> > + int err;
> > +
> > + err = register_filesystem(&devpts_fs_type);
> > + if (err)
> > + return err;
> > +
> > + mnt = kern_mount_data(&devpts_fs_type, NULL);
> > + if (IS_ERR(mnt))
> > + err = PTR_ERR(mnt);
> > + else
> > + devpts_mnt = mnt;
> >   return err;
> > }
> >
> > _____
> > 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

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Pavel Emelianov on Wed, 06 Feb 2008 16:24:17 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> sukadev@us.ibm.com wrote:
>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
>>>
>>> To support multiple PTY namespaces, we should be allow multiple mounts of
>>> /dev/pts, once within each PTY namespace.
>>>
>>> This patch removes the get_sb_single() in devpts_get_sb() and uses test and
>>> set sb interfaces to allow remounting /dev/pts. The patch also removes the
>>> globals, 'devpts_root' and uses current_pts_mnt() to access 'devpts_mnt'
>>>
>>> Changelog:
>>> - Version 0: Based on earlier versions from Serge Hallyn and
>>> Matt Helsley.
>>>
>>> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>> ---
>>> fs/devpts/inode.c | 120 ++---------
>>> 1 file changed, 101 insertions(+), 19 deletions(-)
>>>
>>> Index: linux-2.6.24/fs/devpts/inode.c
>>> ===
>>> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 17:30:52.000000000 -0800
>>> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
>>> @@ -34,7 +34,10 @@ static inline struct idr *current_pts_ns
>>> }
>>>
>>> static struct vfsmount *devpts_mnt;
>>> -static struct dentry *devpts_root;
>>> +static inline struct vfsmount *current_pts_ns_mnt(void)

```
>>> +{
>>> + return devpts_mnt;
>>> +}
>>>
>>>  static struct {
>>>   int setuid;
>>> @@ -130,7 +133,7 @@ devpts_fill_super(struct super_block *s,
>>>   inode->i_fop = &simple_dir_operations;
>>>   inode->i_nlink = 2;
>>>
>>> - devpts_root = s->s_root = d_alloc_root(inode);
>>> + s->s_root = d_alloc_root(inode);
>>>   if (s->s_root)
>>>    return 0;
>>>
>>> @@ -140,10 +143,53 @@ fail:
>>>   return -ENOMEM;
>>>  }
>>>
>>> +/*
>>> + * We use test and set super-block operations to help determine whether we
>>> + * need a new super-block for this namespace. get_sb() walks the list of
>>> + * existing devpts supers, comparing them with the @data ptr. Since we
>>> + * passed 'current's namespace as the @data pointer we can compare the
>>> + * namespace pointer in the super-block's 's_fs_info'.  If the test is
>>> + * TRUE then get_sb() returns a new active reference to the super block.
>>> + * Otherwise, it helps us build an active reference to a new one.
>>> + */
>>> +
>>> +static int devpts_test_sb(struct super_block *sb, void *data)
>>> +{
>>> + return sb->s_fs_info == data;
>>> +}
>>> +
>>> +static int devpts_set_sb(struct super_block *sb, void *data)
>>> +{
>>> + sb->s_fs_info = data;
>>> + return set_anon_super(sb, NULL);
>>> +}
>>> +
>>>  static int devpts_get_sb(struct file_system_type *fs_type,
>>>   int flags, const char *dev_name, void *data, struct vfsmount *mnt)
>>>  {
>>> - return get_sb_single(fs_type, flags, data, devpts_fill_super, mnt);
>>> + struct super_block *sb;
>>> + int err;
>>> +
>>> + /* hereafter we're very simlar to get_sb_nodev */
```

>>> + sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
>>> + if (IS_ERR(sb))
>>> + return PTR_ERR(sb);
>>> +
>>> + if (sb->s_root)
>>> + return simple_set_mnt(mnt, sb);
>>> +
>>> + sb->s_flags = flags;
>>> + err = devpts_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
>>> + if (err) {
>>> + up_write(&sb->s_umount);
>>> + deactivate_super(sb);
>>> + return err;
>>> + }
>>> +
>> That stuff becomes very very similar to that in proc :)
>> Makes sense to consolidate. Maybe...
>
> Yeah, and the mqns that Cedric sent too. I think Cedric said he'd
> started an a patch implementing a helper. Cedric?

Mmm. I wanted to send one small objection to Cedric's patches with mqns,
but the thread was abandoned by the time I decided to do-it-right-now.

So I can put it here: forcing the CLONE_NEWNS is not very good, since
this makes impossible to push a bind mount inside a new namespace, which
may operate in some chroot environment. But this ability is heavily
exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
that would be very very good :) See my next comment about this issue.

> Pavel, not long ago you said you were starting to look at tty and pty
> stuff - did you have any different ideas on devpts virtualization, or
> are you ok with this minus your comments thus far?

I have a similar idea of how to implement this, but I didn't thought
about the details. As far as this issue is concerned, I see no reasons
why we need a kern_mount-ed devtpsfs instance. If we don't make such,
we may safely hold the ptsns from the superblock and be happy. The
same seems applicable to the mqns, no?

The reason I have the kern_mount-ed instance of proc for pid namespaces
is that I need a vfsmount to flush task entries from, but allowing
it to be NULL (i.e. no kern_mount, but optional user mounts) means
handing all the possible races, which is too heavy. But do we actually
need the vfsmount for devpts and mqns if no user-space mounts exist?

Besides, I planned to include legacy ptys virtualization and console
virtualizatin in this namespace, but it seems, that it is not present

in this particular one.

```
>>> + sb->s_flags |= MS_ACTIVE;
>>> + devpts_mnt = mnt;
>>> +
>>> + return simple_set_mnt(mnt, sb);
>>>  }
>>>
>>>  static struct file_system_type devpts_fs_type = {
>>> @@ -158,10 +204,9 @@ static struct file_system_type devpts_fs
>>>   * to the System V naming convention
>>>   */
>>>
>>> -static struct dentry *get_node(int num)
>>> +static struct dentry *get_node(struct dentry *root, int num)
>>>  {
>>>   char s[12];
>>> - struct dentry *root = devpts_root;
>>>   mutex_lock(&root->d_inode->i_mutex);
>>>   return lookup_one_len(s, root, sprintf(s, "%d", num));
>>>  }
>>> @@ -207,12 +252,28 @@ int devpts_pty_new(struct tty_struct *tt
>>>   struct tty_driver *driver = tty->driver;
>>>   dev_t device = MKDEV(driver->major, driver->minor_start+number);
>>>   struct dentry *dentry;
>>> - struct inode *inode = new_inode(devpts_mnt->mnt_sb);
>>> + struct dentry *root;
>>> + struct vfsmount *mnt;
>>> + struct inode *inode;
>>> +
>>>
>>>   /* We're supposed to be given the slave end of a pty */
>>>   BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
>>>   BUG_ON(driver->subtype != PTY_TYPE_SLAVE);
>>>
>>> + mnt = current_pts_ns_mnt();
>>> + if (!mnt)
>>> +  return -ENOSYS;
>>> + root = mnt->mnt_root;
>>> +
>>> + mutex_lock(&root->d_inode->i_mutex);
>>> + inode = idr_find(current_pts_ns_allocated_ptys(), number);
>>> + mutex_unlock(&root->d_inode->i_mutex);
>>> +
>>> + if (inode && !IS_ERR(inode))
>>> +  return -EEXIST;
>>> +
>>> + inode = new_inode(mnt->mnt_sb);
```

```
>>>   if (!inode)
>>>    return -ENOMEM;
>>>
>>> @@ -222,23 +283,31 @@ int devpts_pty_new(struct tty_struct *tt
>>>   inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
>>>   init_special_inode(inode, S_IFCHR|config.mode, device);
>>>   inode->i_private = tty;
>>> + idr_replace(current_pts_ns_allocated_ptys(), inode, number);
>>>
>>> - dentry = get_node(number);
>>> + dentry = get_node(root, number);
>>>   if (!IS_ERR(dentry) && !dentry->d_inode) {
>>>    d_instantiate(dentry, inode);
>>> -  fsnotify_create(devpts_root->d_inode, dentry);
>>> +  fsnotify_create(root->d_inode, dentry);
>>>   }
>>>
>>> - mutex_unlock(&devpts_root->d_inode->i_mutex);
>>> + mutex_unlock(&root->d_inode->i_mutex);
>>>
>>>   return 0;
>>>  }
>>>
>>>  struct tty_struct *devpts_get_tty(int number)
>>>  {
>>> - struct dentry *dentry = get_node(number);
>>> + struct vfsmount *mnt;
>>> + struct dentry *dentry;
>>>   struct tty_struct *tty;
>>>
>>> + mnt = current_pts_ns_mnt();
>>> + if (!mnt)
>>> +  return NULL;
>>> +
>>> + dentry = get_node(mnt->mnt_root, number);
>>> +
>>>   tty = NULL;
>>>   if (!IS_ERR(dentry)) {
>>>    if (dentry->d_inode)
>>> @@ -246,14 +315,21 @@ struct tty_struct *devpts_get_tty(int nu
>>>    dput(dentry);
>>>   }
>>>
>>> - mutex_unlock(&devpts_root->d_inode->i_mutex);
>>> + mutex_unlock(&mnt->mnt_root->d_inode->i_mutex);
>>>
>>>   return tty;
>>>  }
```

```
>>>
>>>  void devpts_pty_kill(int number)
>>>  {
>>> - struct dentry *dentry = get_node(number);
>>> + struct dentry *dentry;
>>> + struct dentry *root;
>>> + struct vfsmount *mnt;
>>> +
>>> + mnt = current_pts_ns_mnt();
>>> + root = mnt->mnt_root;
>>> +
>>> + dentry = get_node(root, number);
>>>
>>>   if (!IS_ERR(dentry)) {
>>>    struct inode *inode = dentry->d_inode;
>>> @@ -264,17 +340,23 @@ void devpts_pty_kill(int number)
>>>    }
>>>    dput(dentry);
>>>   }
>>> - mutex_unlock(&devpts_root->d_inode->i_mutex);
>>> + mutex_unlock(&root->d_inode->i_mutex);
>>>  }
>>>
>>>  static int __init init_devpts_fs(void)
>>>  {
>>> - int err = register_filesystem(&devpts_fs_type);
>>> - if (!err) {
>>> -  devpts_mnt = kern_mount(&devpts_fs_type);
>>> -  if (IS_ERR(devpts_mnt))
>>> -   err = PTR_ERR(devpts_mnt);
>>> - }
>>> + struct vfsmount *mnt;
>>> + int err;
>>> +
>>> + err = register_filesystem(&devpts_fs_type);
>>> + if (err)
>>> +  return err;
>>> +
>>> + mnt = kern_mount_data(&devpts_fs_type, NULL);
>>> + if (IS_ERR(mnt))
>>> +  err = PTR_ERR(mnt);
>>> + else
>>> +  devpts_mnt = mnt;
>>>   return err;
>>>  }
>>>
>>> _____________________________________________________
>>> 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
>

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

---

Subject: Re: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
Posted by serue on Wed, 06 Feb 2008 16:25:57 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >> sukadev@us.ibm.com wrote:
> >>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> >>> Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
> >>>
> >>> Enable cloning PTY namespaces.
> >>>
> >>> TODO:
> >>> This version temporarily uses the clone flag '0x80000000' which
> >>> is unused in mainline atm, but used for CLONE_IO in -mm.
> >>> While we must extend clone() (urgently) to solve this, it hopefully
> >>> does not affect review of the rest of this patchset.
> >>>
> >>> Changelog:
> >>> - Version 0: Based on earlier versions from Serge Hallyn and
> >>> Matt Helsley.
> >>>
> >>> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> >>> ---
> >>> fs/devpts/inode.c | 84 +++-------
> >>> include/linux/devpts_fs.h | 52 ++++++++++++++++++++++++++++++
> >>> include/linux/init_task.h | 1
> >>> include/linux/nsproxy.h | 2 +
> >>> include/linux/sched.h | 2 +
> >>> kernel/fork.c | 2 -
> >>> kernel/nsproxy.c | 17 ++++++++-
> >>> 7 files changed, 146 insertions(+), 14 deletions(-)

```
> >>>
> >>> Index: linux-2.6.24/fs/devpts/inode.c
> >>> ===================================================================
> >>> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
> >>> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 20:27:41.000000000 -0800
> >>> @@ -25,18 +25,25 @@
> >>>  #define DEVPTS_SUPER_MAGIC 0x1cd1
> >>>
> >>>  extern int pty_limit;  /* Config limit on Unix98 ptys */
> >>> -static DEFINE_IDR(allocated_ptys);
> >>>  static DECLARE_MUTEX(allocated_ptys_lock);
> >>> +static struct file_system_type devpts_fs_type;
> >>> +
> >>> +struct pts_namespace init_pts_ns = {
> >>> + .kref = {
> >>> +  .refcount = ATOMIC_INIT(2),
> >>> + },
> >>> + .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
> >>> + .mnt = NULL,
> >>> +};
> >>>
> >>>  static inline struct idr *current_pts_ns_allocated_ptys(void)
> >>>  {
> >>> - return &allocated_ptys;
> >>> + return &current->nsproxy->pts_ns->allocated_ptys;
> >>>  }
> >>>
> >>> -static struct vfsmount *devpts_mnt;
> >>>  static inline struct vfsmount *current_pts_ns_mnt(void)
> >>>  {
> >>> - return devpts_mnt;
> >>> + return current->nsproxy->pts_ns->mnt;
> >>>  }
> >>>
> >>>  static struct {
> >>> @@ -59,6 +66,42 @@ static match_table_t tokens = {
> >>>   {Opt_err, NULL}
> >>>  };
> >>>
> >>> +struct pts_namespace *new_pts_ns(void)
> >>> +{
> >>> + struct pts_namespace *ns;
> >>> +
> >>> + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
> >>> + if (!ns)
> >>> +  return ERR_PTR(-ENOMEM);
> >>> +
> >>> + ns->mnt = kern_mount_data(&devpts_fs_type, ns);
```

> >> You create a circular references here - the namespace
> >> holds the vfsmnt, the vfsmnt holds a superblock, a superblock
> >> holds the namespace.
> >
> > Hmm, yeah, good point. That was probably in my original version last
> > year, so my fault not Suka's. Suka, would it work to have the
> > sb->s_info point to the namespace but not grab a reference, than have
>
> If you don't then you may be in situation, when this devpts
> is mounted from userspace and in case the namespace is dead
> superblock will point to garbage... Superblock MUST hold the
> namespace :)

But when the ns is freed sb->s_info would be NULL. Surely the helpers
can be made to handle that safely?

>
> > free_pts_ns() null out its sb->s_info, i.e. something like
> >
> > void free_pts_ns(struct kref *ns_kref)
> > {
> >   struct pts_namespace *ns;
> >   struct super_block *sb;
> >
> >   ns = container_of(ns_kref, struct pts_namespace, kref);
> >   BUG_ON(ns == &init_pts_ns);
> >   sb = ns->mnt->mnt_sb;
> >
> >   mntput(ns->mnt);
> >   sb->s_info = NULL;
> >
> >   /*
> >   * TODO:
> >   *   idr_remove_all(&ns->allocated_ptys); introduced in
> > .6.23
> >   */
> >   idr_destroy(&ns->allocated_ptys);
> >   kfree(ns);
> > }
> >
> >

Subject: Re: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
Posted by Pavel Emelianov on Wed, 06 Feb 2008 16:35:18 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> Serge E. Hallyn wrote:
>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>> sukadev@us.ibm.com wrote:
>>>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>>>> Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
>>>>>
>>>>> Enable cloning PTY namespaces.
>>>>>
>>>>> TODO:
>>>>> This version temporarily uses the clone flag '0x80000000' which
>>>>> is unused in mainline atm, but used for CLONE_IO in -mm.
>>>>> While we must extend clone() (urgently) to solve this, it hopefully
>>>>> does not affect review of the rest of this patchset.
>>>>>
>>>>> Changelog:
>>>>> - Version 0: Based on earlier versions from Serge Hallyn and
>>>>> Matt Helsley.
>>>>>
>>>>> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>>>> ---
>>>>> fs/devpts/inode.c | 84 +++++++++++++++++++++++++++++++++++++++-------
>>>>> include/linux/devpts_fs.h | 52 ++++++++++++++++++++++++++++++
>>>>> include/linux/init_task.h | 1
>>>>> include/linux/nsproxy.h | 2 +
>>>>> include/linux/sched.h | 2 +
>>>>> kernel/fork.c | 2 -
>>>>> kernel/nsproxy.c | 17 ++++++++-
>>>>> 7 files changed, 146 insertions(+), 14 deletions(-)
>>>>>
>>>>> Index: linux-2.6.24/fs/devpts/inode.c
>>>>> ===
>>>>> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
>>>>> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 20:27:41.000000000 -0800
>>>>> @@ -25,18 +25,25 @@
>>>>> #define DEVPTS_SUPER_MAGIC 0x1cd1
>>>>>
>>>>> extern int pty_limit; /* Config limit on Unix98 ptys */
>>>>> -static DEFINE_IDR(allocated_ptys);
>>>>> static DECLARE_MUTEX(allocated_ptys_lock);
>>>>> +static struct file_system_type devpts_fs_type;
>>>>> +
>>>>> +struct pts_namespace init_pts_ns = {
>>>>> + .kref = {

>>>>> + .refcount = ATOMIC_INIT(2),
>>>>> + },
>>>>> + .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
>>>>> + .mnt = NULL,
>>>>> +};
>>>>>
>>>>> static inline struct idr *current_pts_ns_allocated_ptys(void)
>>>>> {
>>>>> - return &allocated_ptys;
>>>>> + return &current->nsproxy->pts_ns->allocated_ptys;
>>>>> }
>>>>>
>>>>> -static struct vfsmount *devpts_mnt;
>>>>> static inline struct vfsmount *current_pts_ns_mnt(void)
>>>>> {
>>>>> - return devpts_mnt;
>>>>> + return current->nsproxy->pts_ns->mnt;
>>>>> }
>>>>>
>>>>> static struct {
>>>>> @@ -59,6 +66,42 @@ static match_table_t tokens = {
>>>>>   {Opt_err, NULL}
>>>>> };
>>>>>
>>>>> +struct pts_namespace *new_pts_ns(void)
>>>>> +{
>>>>> + struct pts_namespace *ns;
>>>>> +
>>>>> + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
>>>>> + if (!ns)
>>>>> + return ERR_PTR(-ENOMEM);
>>>>> +
>>>>> + ns->mnt = kern_mount_data(&devpts_fs_type, ns);
>>>> You create a circular references here - the namespace
>>>> holds the vfsmnt, the vfsmnt holds a superblock, a superblock
>>>> holds the namespace.
>>> Hmm, yeah, good point. That was probably in my original version last
>>> year, so my fault not Suka's. Suka, would it work to have the
>>> sb->s_info point to the namespace but not grab a reference, than have
>> If you don't then you may be in situation, when this devpts
>> is mounted from userspace and in case the namespace is dead
>> superblock will point to garbage... Superblock MUST hold the
>> namespace :)
>
> But when the ns is freed sb->s_info would be NULL. Surely the helpers
> can be made to handle that safely?

Hm... How do we find the proper superblock? Have a reference on

it from the namespace? I'm afraid it will be easy to resolve the
locking issues here.

I propose another scheme - we simply don't have ANY references
from namespace to superblock/vfsmount, but get the current
namespace in devpts_get_sb() and put in devpts_free_sb().

```
>>> free_pts_ns() null out its sb->s_info, i.e. something like
>>>
>>> void free_pts_ns(struct kref *ns_kref)
>>> {
>>>    struct pts_namespace *ns;
>>>    struct super_block *sb;
>>>
>>>    ns = container_of(ns_kref, struct pts_namespace, kref);
>>>    BUG_ON(ns == &init_pts_ns);
>>>    sb = ns->mnt->mnt_sb;
>>>
>>>    mntput(ns->mnt);
>>>    sb->s_info = NULL;
>>>
>>>    /*
>>>     * TODO:
>>>     *      idr_remove_all(&ns->allocated_ptys); introduced in
>>> .6.23
>>>     */
>>>    idr_destroy(&ns->allocated_ptys);
>>>    kfree(ns);
>>> }
>>>
>>>
>
```

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by serue on Wed, 06 Feb 2008 16:43:28 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >> sukadev@us.ibm.com wrote:
> >>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

> >>> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
> >>>
> >>> To support multiple PTY namespaces, we should be allow multiple mounts of
> >>> /dev/pts, once within each PTY namespace.
> >>>
> >>> This patch removes the get_sb_single() in devpts_get_sb() and uses test and
> >>> set sb interfaces to allow remounting /dev/pts. The patch also removes the
> >>> globals, 'devpts_root' and uses current_pts_mnt() to access 'devpts_mnt'
> >>>
> >>> Changelog:
> >>> - Version 0: Based on earlier versions from Serge Hallyn and
> >>>   Matt Helsley.
> >>>
> >>> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> >>> ---
> >>> fs/devpts/inode.c | 120 ++---------
> >>> 1 file changed, 101 insertions(+), 19 deletions(-)
> >>>
> >>> Index: linux-2.6.24/fs/devpts/inode.c
> >>> ===
> >>> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 17:30:52.000000000 -0800
> >>> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
> >>> @@ -34,7 +34,10 @@ static inline struct idr *current_pts_ns
> >>> }
> >>>
> >>> static struct vfsmount *devpts_mnt;
> >>> -static struct dentry *devpts_root;
> >>> +static inline struct vfsmount *current_pts_ns_mnt(void)
> >>> +{
> >>> + return devpts_mnt;
> >>> +}
> >>>
> >>> static struct {
> >>>   int setuid;
> >>> @@ -130,7 +133,7 @@ devpts_fill_super(struct super_block *s,
> >>>   inode->i_fop = &simple_dir_operations;
> >>>   inode->i_nlink = 2;
> >>>
> >>> - devpts_root = s->s_root = d_alloc_root(inode);
> >>> + s->s_root = d_alloc_root(inode);
> >>>   if (s->s_root)
> >>>   return 0;
> >>>
> >>> @@ -140,10 +143,53 @@ fail:
> >>>   return -ENOMEM;
> >>> }
> >>>
> >>> +/*

> >>> + * We use test and set super-block operations to help determine whether we
> >>> + * need a new super-block for this namespace. get_sb() walks the list of
> >>> + * existing devpts supers, comparing them with the @data ptr. Since we
> >>> + * passed 'current's namespace as the @data pointer we can compare the
> >>> + * namespace pointer in the super-block's 's_fs_info'. If the test is
> >>> + * TRUE then get_sb() returns a new active reference to the super block.
> >>> + * Otherwise, it helps us build an active reference to a new one.
> >>> + */
> >>> +
> >>> +static int devpts_test_sb(struct super_block *sb, void *data)
> >>> +{
> >>> + return sb->s_fs_info == data;
> >>> +}
> >>> +
> >>> +static int devpts_set_sb(struct super_block *sb, void *data)
> >>> +{
> >>> + sb->s_fs_info = data;
> >>> + return set_anon_super(sb, NULL);
> >>> +}
> >>> +
> >>> static int devpts_get_sb(struct file_system_type *fs_type,
> >>>   int flags, const char *dev_name, void *data, struct vfsmount *mnt)
> >>> {
> >>> - return get_sb_single(fs_type, flags, data, devpts_fill_super, mnt);
> >>> + struct super_block *sb;
> >>> + int err;
> >>> +
> >>> + /* hereafter we're very simlar to get_sb_nodev */
> >>> + sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
> >>> + if (IS_ERR(sb))
> >>> + return PTR_ERR(sb);
> >>> +
> >>> + if (sb->s_root)
> >>> + return simple_set_mnt(mnt, sb);
> >>> +
> >>> + sb->s_flags = flags;
> >>> + err = devpts_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
> >>> + if (err) {
> >>> + up_write(&sb->s_umount);
> >>> + deactivate_super(sb);
> >>> + return err;
> >>> + }
> >>> +
> >> That stuff becomes very very similar to that in proc :)
> >> Makes sense to consolidate. Maybe...
> >
> > Yeah, and the mqns that Cedric sent too. I think Cedric said he'd
> > started an a patch implementing a helper. Cedric?

>
> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
> but the thread was abandoned by the time I decided to do-it-right-now.
>
> So I can put it here: forcing the CLONE_NEWNS is not very good, since
> this makes impossible to push a bind mount inside a new namespace, which
> may operate in some chroot environment. But this ability is heavily

Which direction do you want to go? I'm wondering whether mounts
propagation can address it.

Though really, I think you're right - we shouldn't break the kernel
doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
force the combination.

> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
> that would be very very good :) See my next comment about this issue.
>
> > Pavel, not long ago you said you were starting to look at tty and pty
> > stuff - did you have any different ideas on devpts virtualization, or
> > are you ok with this minus your comments thus far?
>
> I have a similar idea of how to implement this, but I didn't thought
> about the details. As far as this issue is concerned, I see no reasons
> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
> we may safely hold the ptsns from the superblock and be happy. The
> same seems applicable to the mqns, no?

But the current->nsproxy->devpts->mnt is used in several functions in
patch 3.

> The reason I have the kern_mount-ed instance of proc for pid namespaces
> is that I need a vfsmount to flush task entries from, but allowing
> it to be NULL (i.e. no kern_mount, but optional user mounts) means
> handing all the possible races, which is too heavy. But do we actually
> need the vfsmount for devpts and mqns if no user-space mounts exist?
>
> Besides, I planned to include legacy ptys virtualization and console
> virtualizatin in this namespace, but it seems, that it is not present
> in this particular one.

I had been thinking the consoles would have their own ns, since there's
really nothing linking them, but there really is no good reason why
userspace should ever want them separate. So I'm fine with combining
them.

> >>> + sb->s_flags |= MS_ACTIVE;
> >>> + devpts_mnt = mnt;

```
> >>> +
> >>> + return simple_set_mnt(mnt, sb);
> >>>  }
> >>>
> >>>  static struct file_system_type devpts_fs_type = {
> >>> @@ -158,10 +204,9 @@ static struct file_system_type devpts_fs
> >>>   * to the System V naming convention
> >>>   */
> >>>
> >>> -static struct dentry *get_node(int num)
> >>> +static struct dentry *get_node(struct dentry *root, int num)
> >>>  {
> >>>   char s[12];
> >>> - struct dentry *root = devpts_root;
> >>>   mutex_lock(&root->d_inode->i_mutex);
> >>>   return lookup_one_len(s, root, sprintf(s, "%d", num));
> >>>  }
> >>> @@ -207,12 +252,28 @@ int devpts_pty_new(struct tty_struct *tt
> >>>   struct tty_driver *driver = tty->driver;
> >>>   dev_t device = MKDEV(driver->major, driver->minor_start+number);
> >>>   struct dentry *dentry;
> >>> - struct inode *inode = new_inode(devpts_mnt->mnt_sb);
> >>> + struct dentry *root;
> >>> + struct vfsmount *mnt;
> >>> + struct inode *inode;
> >>> +
> >>>
> >>>   /* We're supposed to be given the slave end of a pty */
> >>>   BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
> >>>   BUG_ON(driver->subtype != PTY_TYPE_SLAVE);
> >>>
> >>> + mnt = current_pts_ns_mnt();
> >>> + if (!mnt)
> >>> +  return -ENOSYS;
> >>> + root = mnt->mnt_root;
> >>> +
> >>> + mutex_lock(&root->d_inode->i_mutex);
> >>> + inode = idr_find(current_pts_ns_allocated_ptys(), number);
> >>> + mutex_unlock(&root->d_inode->i_mutex);
> >>> +
> >>> + if (inode && !IS_ERR(inode))
> >>> +  return -EEXIST;
> >>> +
> >>> + inode = new_inode(mnt->mnt_sb);
> >>>   if (!inode)
> >>>    return -ENOMEM;
> >>>
> >>> @@ -222,23 +283,31 @@ int devpts_pty_new(struct tty_struct *tt
```

```
> >>>   inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
> >>>   init_special_inode(inode, S_IFCHR|config.mode, device);
> >>>   inode->i_private = tty;
> >>> + idr_replace(current_pts_ns_allocated_ptys(), inode, number);
> >>>
> >>> - dentry = get_node(number);
> >>> + dentry = get_node(root, number);
> >>>   if (!IS_ERR(dentry) && !dentry->d_inode) {
> >>>    d_instantiate(dentry, inode);
> >>> -  fsnotify_create(devpts_root->d_inode, dentry);
> >>> +  fsnotify_create(root->d_inode, dentry);
> >>>   }
> >>>
> >>> - mutex_unlock(&devpts_root->d_inode->i_mutex);
> >>> + mutex_unlock(&root->d_inode->i_mutex);
> >>>
> >>>   return 0;
> >>>  }
> >>>
> >>>  struct tty_struct *devpts_get_tty(int number)
> >>>  {
> >>> - struct dentry *dentry = get_node(number);
> >>> + struct vfsmount *mnt;
> >>> + struct dentry *dentry;
> >>>   struct tty_struct *tty;
> >>>
> >>> + mnt = current_pts_ns_mnt();
> >>> + if (!mnt)
> >>> +  return NULL;
> >>> +
> >>> + dentry = get_node(mnt->mnt_root, number);
> >>> +
> >>>   tty = NULL;
> >>>   if (!IS_ERR(dentry)) {
> >>>    if (dentry->d_inode)
> >>> @@ -246,14 +315,21 @@ struct tty_struct *devpts_get_tty(int nu
> >>>    dput(dentry);
> >>>   }
> >>>
> >>> - mutex_unlock(&devpts_root->d_inode->i_mutex);
> >>> + mutex_unlock(&mnt->mnt_root->d_inode->i_mutex);
> >>>
> >>>   return tty;
> >>>  }
> >>>
> >>>  void devpts_pty_kill(int number)
> >>>  {
> >>> - struct dentry *dentry = get_node(number);
```

> >>> + struct dentry *dentry;
> >>> + struct dentry *root;
> >>> + struct vfsmount *mnt;
> >>> +
> >>> + mnt = current_pts_ns_mnt();
> >>> + root = mnt->mnt_root;
> >>> +
> >>> + dentry = get_node(root, number);
> >>>
> >>> if (!IS_ERR(dentry)) {
> >>> struct inode *inode = dentry->d_inode;
> >>> @@ -264,17 +340,23 @@ void devpts_pty_kill(int number)
> >>> }
> >>> dput(dentry);
> >>> }
> >>> - mutex_unlock(&devpts_root->d_inode->i_mutex);
> >>> + mutex_unlock(&root->d_inode->i_mutex);
> >>> }
> >>>
> >>> static int __init init_devpts_fs(void)
> >>> {
> >>> - int err = register_filesystem(&devpts_fs_type);
> >>> - if (!err) {
> >>> - devpts_mnt = kern_mount(&devpts_fs_type);
> >>> - if (IS_ERR(devpts_mnt))
> >>> - err = PTR_ERR(devpts_mnt);
> >>> - }
> >>> + struct vfsmount *mnt;
> >>> + int err;
> >>> +
> >>> + err = register_filesystem(&devpts_fs_type);
> >>> + if (err)
> >>> + return err;
> >>> +
> >>> + mnt = kern_mount_data(&devpts_fs_type, NULL);
> >>> + if (IS_ERR(mnt))
> >>> + err = PTR_ERR(mnt);
> >>> + else
> >>> + devpts_mnt = mnt;
> >>> return err;
> >>> }
> >>>
> >>> _____
> >>> 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
> >

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Pavel Emelianov on Wed, 06 Feb 2008 16:56:37 GMT
View Forum Message <> Reply to Message

[snip]

>> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
>> but the thread was abandoned by the time I decided to do-it-right-now.
>>
>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
>> this makes impossible to push a bind mount inside a new namespace, which
>> may operate in some chroot environment. But this ability is heavily
>
> Which direction do you want to go? I'm wondering whether mounts
> propagation can address it.

Hardly. AFAIS there's no way to let the chroot-ed tasks see parts of
vfs tree, that left behind them after chroot, unless they are in the
same mntns as you, and you bind mount this parts to their tree. No?

> Though really, I think you're right - we shouldn't break the kernel
> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
> force the combination.
>
>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
>> that would be very very good :) See my next comment about this issue.
>>
>>> Pavel, not long ago you said you were starting to look at tty and pty
>>> stuff - did you have any different ideas on devpts virtualization, or
>>> are you ok with this minus your comments thus far?
>> I have a similar idea of how to implement this, but I didn't thought
>> about the details. As far as this issue is concerned, I see no reasons
>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
>> we may safely hold the ptsns from the superblock and be happy. The
>> same seems applicable to the mqns, no?
>
> But the current->nsproxy->devpts->mnt is used in several functions in
> patch 3.

Indeed. I overlooked this. Then we're in a deep ... problem here.

Breaking this circle was not that easy with pid namespaces, so
I put the strut in proc_flush_task - when the last task from the
namespace exits the kern-mount-ed vfsmnt is dropped, but we can't
do the same stuff with devpts.

I do not remember now what the problem was and it's already quite
late in Moscow, so if you don't mind I'll revisit the issue tomorrow.

Off-topic: does any of you know whether Andrew is willing to accept
new features in the nearest future? The problem is that I have a
device visibility controller fixed and pending to send, but I can't
guess a good time for it :)

>> The reason I have the kern_mount-ed instance of proc for pid namespaces
>> is that I need a vfsmount to flush task entries from, but allowing
>> it to be NULL (i.e. no kern_mount, but optional user mounts) means
>> handing all the possible races, which is too heavy. But do we actually
>> need the vfsmount for devpts and mqns if no user-space mounts exist?
>>
>> Besides, I planned to include legacy ptys virtualization and console
>> virtualizatin in this namespace, but it seems, that it is not present
>> in this particular one.
>
> I had been thinking the consoles would have their own ns, since there's
> really nothing linking them, but there really is no good reason why
> userspace should ever want them separate. So I'm fine with combining
> them.

OK.

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

---

Subject: Re: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
Posted by serue on Wed, 06 Feb 2008 17:04:02 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >> Serge E. Hallyn wrote:
> >>> Quoting Pavel Emelyanov (xemul@openvz.org):
> >>>> sukadev@us.ibm.com wrote:

> >>>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> >>>>> Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
> >>>>>
> >>>>> Enable cloning PTY namespaces.
> >>>>>
> >>>>> TODO:
> >>>>> This version temporarily uses the clone flag '0x80000000' which
> >>>>> is unused in mainline atm, but used for CLONE_IO in -mm.
> >>>>> While we must extend clone() (urgently) to solve this, it hopefully
> >>>>> does not affect review of the rest of this patchset.
> >>>>>
> >>>>> Changelog:
> >>>>> - Version 0: Based on earlier versions from Serge Hallyn and
> >>>>>   Matt Helsley.
> >>>>>
> >>>>> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> >>>>> ---
> >>>>> fs/devpts/inode.c | 84 ++++++++++++++++++++++++++++++++++++-------
> >>>>> include/linux/devpts_fs.h | 52 ++++++++++++++++++++++++++++
> >>>>> include/linux/init_task.h | 1
> >>>>> include/linux/nsproxy.h | 2 +
> >>>>> include/linux/sched.h | 2 +
> >>>>> kernel/fork.c | 2 -
> >>>>> kernel/nsproxy.c | 17 ++++++++-
> >>>>> 7 files changed, 146 insertions(+), 14 deletions(-)
> >>>>>
> >>>>> Index: linux-2.6.24/fs/devpts/inode.c
> >>>>> ===
> >>>>> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
> >>>>> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 20:27:41.000000000 -0800
> >>>>> @@ -25,18 +25,25 @@
> >>>>> #define DEVPTS_SUPER_MAGIC 0x1cd1
> >>>>>
> >>>>> extern int pty_limit; /* Config limit on Unix98 ptys */
> >>>>> -static DEFINE_IDR(allocated_ptys);
> >>>>> static DECLARE_MUTEX(allocated_ptys_lock);
> >>>>> +static struct file_system_type devpts_fs_type;
> >>>>> +
> >>>>> +struct pts_namespace init_pts_ns = {
> >>>>> + .kref = {
> >>>>> + .refcount = ATOMIC_INIT(2),
> >>>>> + },
> >>>>> + .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
> >>>>> + .mnt = NULL,
> >>>>> +};
> >>>>>
> >>>>> static inline struct idr *current_pts_ns_allocated_ptys(void)
> >>>>> {

> >>>>> - return &allocated_ptys;
> >>>>> + return &current->nsproxy->pts_ns->allocated_ptys;
> >>>>> }
> >>>>>
> >>>>> -static struct vfsmount *devpts_mnt;
> >>>>> static inline struct vfsmount *current_pts_ns_mnt(void)
> >>>>> {
> >>>>> - return devpts_mnt;
> >>>>> + return current->nsproxy->pts_ns->mnt;
> >>>>> }
> >>>>>
> >>>>> static struct {
> >>>>> @@ -59,6 +66,42 @@ static match_table_t tokens = {
> >>>>>   {Opt_err, NULL}
> >>>>> };
> >>>>>
> >>>>> +struct pts_namespace *new_pts_ns(void)
> >>>>> +{
> >>>>> + struct pts_namespace *ns;
> >>>>> +
> >>>>> + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
> >>>>> + if (!ns)
> >>>>> + return ERR_PTR(-ENOMEM);
> >>>>> +
> >>>>> + ns->mnt = kern_mount_data(&devpts_fs_type, ns);
> >>>> You create a circular references here - the namespace
> >>>> holds the vfsmnt, the vfsmnt holds a superblock, a superblock
> >>>> holds the namespace.
> >>> Hmm, yeah, good point. That was probably in my original version last
> >>> year, so my fault not Suka's. Suka, would it work to have the
> >>> sb->s_info point to the namespace but not grab a reference, than have
> >> If you don't then you may be in situation, when this devpts
> >> is mounted from userspace and in case the namespace is dead
> >> superblock will point to garbage... Superblock MUST hold the
> >> namespace :)
> >
> > But when the ns is freed sb->s_info would be NULL. Surely the helpers
> > can be made to handle that safely?
>
> Hm... How do we find the proper superblock? Have a reference on
> it from the namespace? I'm afraid it will be easy to resolve the
> locking issues here.
>
> I propose another scheme - we simply don't have ANY references
> from namespace to superblock/vfsmount, but get the current
> namespace in devpts_get_sb() and put in devpts_free_sb().

But then it really does become impossible to use a /dev/pts from another

namespace, right?

> >>> free_pts_ns() null out its sb->s_info, i.e. something like
> >>>
> >>> void free_pts_ns(struct kref *ns_kref)
> >>> {
> >>> struct pts_namespace *ns;
> >>> struct super_block *sb;
> >>>
> >>> ns = container_of(ns_kref, struct pts_namespace, kref);
> >>> BUG_ON(ns == &init_pts_ns);
> >>> sb = ns->mnt->mnt_sb;
> >>>
> >>> mntput(ns->mnt);
> >>> sb->s_info = NULL;
> >>>
> >>> /*
> >>> * TODO:
> >>> * idr_remove_all(&ns->allocated_ptys); introduced in
> >>> .6.23
> >>> */
> >>> idr_destroy(&ns->allocated_ptys);
> >>> kfree(ns);
> >>> }
> >>>
> >>>
> > >

Subject: Re: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
Posted by Pavel Emelianov on Wed, 06 Feb 2008 17:06:15 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> Serge E. Hallyn wrote:
>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>> Serge E. Hallyn wrote:
>>>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>>>> sukadev@us.ibm.com wrote:
>>>>>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>>>>>> Subject: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
>>>>>>>
>>>>>>> Enable cloning PTY namespaces.

>>>>>>>
>>>>>>> TODO:
>>>>>>> This version temporarily uses the clone flag '0x80000000' which
>>>>>>> is unused in mainline atm, but used for CLONE_IO in -mm.
>>>>>>> While we must extend clone() (urgently) to solve this, it hopefully
>>>>>>> does not affect review of the rest of this patchset.
>>>>>>>
>>>>>>> Changelog:
>>>>>>> - Version 0: Based on earlier versions from Serge Hallyn and
>>>>>>>   Matt Helsley.
>>>>>>>
>>>>>>> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>>>>>> ---
>>>>>>> fs/devpts/inode.c | 84 ++++++++++++++++++++++++++++++++++++-------
>>>>>>> include/linux/devpts_fs.h | 52 ++++++++++++++++++++++++++++
>>>>>>> include/linux/init_task.h | 1
>>>>>>> include/linux/nsproxy.h | 2 +
>>>>>>> include/linux/sched.h | 2 +
>>>>>>> kernel/fork.c | 2 -
>>>>>>> kernel/nsproxy.c | 17 ++++++++-
>>>>>>> 7 files changed, 146 insertions(+), 14 deletions(-)
>>>>>>>
>>>>>>> Index: linux-2.6.24/fs/devpts/inode.c
>>>>>>>
>>>>>>> ===
>>>>>>> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
>>>>>>> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 20:27:41.000000000 -0800
>>>>>>> @@ -25,18 +25,25 @@
>>>>>>> #define DEVPTS_SUPER_MAGIC 0x1cd1
>>>>>>>
>>>>>>> extern int pty_limit; /* Config limit on Unix98 ptys */
>>>>>>> -static DEFINE_IDR(allocated_ptys);
>>>>>>> static DECLARE_MUTEX(allocated_ptys_lock);
>>>>>>> +static struct file_system_type devpts_fs_type;
>>>>>>> +
>>>>>>> +struct pts_namespace init_pts_ns = {
>>>>>>> + .kref = {
>>>>>>> + .refcount = ATOMIC_INIT(2),
>>>>>>> + },
>>>>>>> + .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
>>>>>>> + .mnt = NULL,
>>>>>>> +};
>>>>>>>
>>>>>>> static inline struct idr *current_pts_ns_allocated_ptys(void)
>>>>>>> {
>>>>>>> - return &allocated_ptys;
>>>>>>> + return &current->nsproxy->pts_ns->allocated_ptys;
>>>>>>> }

>>>>>>>
>>>>>>> -static struct vfsmount *devpts_mnt;
>>>>>>> static inline struct vfsmount *current_pts_ns_mnt(void)
>>>>>>> {
>>>>>>> - return devpts_mnt;
>>>>>>> + return current->nsproxy->pts_ns->mnt;
>>>>>>> }
>>>>>>>
>>>>>>> static struct {
>>>>>>> @@ -59,6 +66,42 @@ static match_table_t tokens = {
>>>>>>>   {Opt_err, NULL}
>>>>>>> };
>>>>>>>
>>>>>>> +struct pts_namespace *new_pts_ns(void)
>>>>>>> +{
>>>>>>> + struct pts_namespace *ns;
>>>>>>> +
>>>>>>> + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
>>>>>>> + if (!ns)
>>>>>>> + return ERR_PTR(-ENOMEM);
>>>>>>> +
>>>>>>> + ns->mnt = kern_mount_data(&devpts_fs_type, ns);
>>>>>> You create a circular references here - the namespace
>>>>>> holds the vfsmnt, the vfsmnt holds a superblock, a superblock
>>>>>> holds the namespace.
>>>>> Hmm, yeah, good point. That was probably in my original version last
>>>>> year, so my fault not Suka's. Suka, would it work to have the
>>>>> sb->s_info point to the namespace but not grab a reference, than have
>>>> If you don't then you may be in situation, when this devpts
>>>> is mounted from userspace and in case the namespace is dead
>>>> superblock will point to garbage... Superblock MUST hold the
>>>> namespace :)
>>> But when the ns is freed sb->s_info would be NULL. Surely the helpers
>>> can be made to handle that safely?
>> Hm... How do we find the proper superblock? Have a reference on
>> it from the namespace? I'm afraid it will be easy to resolve the
>> locking issues here.
>>
>> I propose another scheme - we simply don't have ANY references
>> from namespace to superblock/vfsmount, but get the current
>> namespace in devpts_get_sb() and put in devpts_free_sb().
>
> But then it really does become impossible to use a /dev/pts from another
> namespace, right?

Right. I already see this from another thread :) Let's drop this one.

>>>>> free_pts_ns() null out its sb->s_info, i.e. something like

```
>>>>>
>>>>> void free_pts_ns(struct kref *ns_kref)
>>>>> {
>>>>>    struct pts_namespace *ns;
>>>>>    struct super_block *sb;
>>>>>
>>>>>    ns = container_of(ns_kref, struct pts_namespace, kref);
>>>>>    BUG_ON(ns == &init_pts_ns);
>>>>>    sb = ns->mnt->mnt_sb;
>>>>>
>>>>>    mntput(ns->mnt);
>>>>>    sb->s_info = NULL;
>>>>>
>>>>>    /*
>>>>>     * TODO:
>>>>>     *      idr_remove_all(&ns->allocated_ptys); introduced in
>>>>> .6.23
>>>>>     */
>>>>>    idr_destroy(&ns->allocated_ptys);
>>>>>    kfree(ns);
>>>>> }
>>>>>
>>>>>
 >
```

_____

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by serue on Wed, 06 Feb 2008 17:32:11 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> [snip]
>
> >> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
> >> but the thread was abandoned by the time I decided to do-it-right-now.
> >>
> >> So I can put it here: forcing the CLONE_NEWNS is not very good, since
> >> this makes impossible to push a bind mount inside a new namespace, which
> >> may operate in some chroot environment. But this ability is heavily
> >
> > Which direction do you want to go? I'm wondering whether mounts
> > propagation can address it.
>

> Hardly. AFAIS there's no way to let the chroot-ed tasks see parts of
> vfs tree, that left behind them after chroot, unless they are in the
> same mntns as you, and you bind mount this parts to their tree. No?

Well no, but I suspect I'm just not understanding what you want to do.
But if the chroot is under /jail1, and you've done, say,

```
mkdir -p /share/pts
mkdir -p /jail1/share
mount --bind /share /share
mount --make-shared /share
mount --bind /share /jail1/share
mount --make-slave /jail1/share
```

before the chroot-ed tasks were cloned with CLONE_NEWNS, then when you
do

```
mount --bind /dev/pts /share/pts
```

from the parent mntns (not that I know why you'd want to do *that* :)
then the chroot'ed tasks will see the original mntns's /dev/pts under
/jail1/share.

>
> > Though really, I think you're right - we shouldn't break the kernel
> > doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
> > force the combination.
> >
> >> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
> >> that would be very very good :) See my next comment about this issue.
> >>
> >>> Pavel, not long ago you said you were starting to look at tty and pty
> >>> stuff - did you have any different ideas on devpts virtualization, or
> >>> are you ok with this minus your comments thus far?
> >> I have a similar idea of how to implement this, but I didn't thought
> >> about the details. As far as this issue is concerned, I see no reasons
> >> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
> >> we may safely hold the ptsns from the superblock and be happy. The
> >> same seems applicable to the mqns, no?
> >
> > But the current->nsproxy->devpts->mnt is used in several functions in
> > patch 3.
>
> Indeed. I overlooked this. Then we're in a deep ... problem here.
>
> Breaking this circle was not that easy with pid namespaces, so
> I put the strut in proc_flush_task - when the last task from the
> namespace exits the kern-mount-ed vfsmnt is dropped, but we can't

> do the same stuff with devpts.

But I still don't see what the problem is with my proposal? So long as
you agree that if there are no tasks remaining in the devptsns,
then any task which has its devpts mounted should see an empty directory
(due to sb->s_info being NULL), I think it works.

>
> I do not remember now what the problem was and it's already quite
> late in Moscow, so if you don't mind I'll revisit the issue tomorrow.

Ok, that's fine. I'll let it sit until then too :) Good night.

> Off-topic: does any of you know whether Andrew is willing to accept
> new features in the nearest future? The problem is that I have a
> device visibility controller fixed and pending to send, but I can't
> guess a good time for it :)

Well even if Andrew won't take it I'd like to see it, so I'd appreciate
a resend.

> >> The reason I have the kern_mount-ed instance of proc for pid namespaces
> >> is that I need a vfsmount to flush task entries from, but allowing
> >> it to be NULL (i.e. no kern_mount, but optional user mounts) means
> >> handing all the possible races, which is too heavy. But do we actually
> >> need the vfsmount for devpts and mqns if no user-space mounts exist?
> >>
> >> Besides, I planned to include legacy ptys virtualization and console
> >> virtualizatin in this namespace, but it seems, that it is not present
> >> in this particular one.
> >
> > I had been thinking the consoles would have their own ns, since there's
> > really nothing linking them, but there really is no good reason why
> > userspace should ever want them separate. So I'm fine with combining
> > them.
>
> OK.

Subject: Re: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
Posted by Cedric Le Goater on Wed, 06 Feb 2008 18:00:22 GMT
View Forum Message <> Reply to Message

>>>>>>

>>>>>> +struct pts_namespace *new_pts_ns(void)
>>>>>> +{
>>>>>> + struct pts_namespace *ns;
>>>>>> +
>>>>>> + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
>>>>>> + if (!ns)
>>>>>> + return ERR_PTR(-ENOMEM);
>>>>>> +
>>>>>> + ns->mnt = kern_mount_data(&devpts_fs_type, ns);
>>>>> You create a circular references here - the namespace
>>>>> holds the vfsmnt, the vfsmnt holds a superblock, a superblock
>>>>> holds the namespace.
>>>> Hmm, yeah, good point. That was probably in my original version last
>>>> year, so my fault not Suka's. Suka, would it work to have the
>>>> sb->s_info point to the namespace but not grab a reference, than have
>>> If you don't then you may be in situation, when this devpts
>>> is mounted from userspace and in case the namespace is dead
>>> superblock will point to garbage... Superblock MUST hold the
>>> namespace :)
>> But when the ns is freed sb->s_info would be NULL. Surely the helpers
>> can be made to handle that safely?
>
> Hm... How do we find the proper superblock? Have a reference on
> it from the namespace? I'm afraid it will be easy to resolve the
> locking issues here.
>
> I propose another scheme - we simply don't have ANY references
> from namespace to superblock/vfsmount, but get the current
> namespace in devpts_get_sb() and put in devpts_free_sb().

I've choosen another path in mq_ns.

I also don't take any refcount on superblock/vfsmount of the new mq_ns
bc of the circular ref. I've considered that namespaces only apply to
processes : the refcount of a namespace is incremented each time a new
task is cloned and the namespace (in my case mq_ns) is released when
the last tasks exists. But this becomes an issue with user mounts which
survives task death. you end up having a user mount pointing to a bogus
mq_ns.

unless you require to have CLONE_NEWNS at the sametime.

Now, this CLONE_NEWNS enforcement seems to be an issue with bind mount.

... jumping to the other thread :)

C.

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Cedric Le Goater on Wed, 06 Feb 2008 18:05:42 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> sukadev@us.ibm.com wrote:
>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
>>>
>>> To support multiple PTY namespaces, we should be allow multiple mounts of
>>> /dev/pts, once within each PTY namespace.
>>>
>>> This patch removes the get_sb_single() in devpts_get_sb() and uses test and
>>> set sb interfaces to allow remounting /dev/pts. The patch also removes the
>>> globals, 'devpts_root' and uses current_pts_mnt() to access 'devpts_mnt'
>>>
>>> Changelog:
>>> - Version 0: Based on earlier versions from Serge Hallyn and
>>> Matt Helsley.
>>>
>>> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>> ---
>>> fs/devpts/inode.c | 120 ++---------
>>> 1 file changed, 101 insertions(+), 19 deletions(-)
>>>
>>> Index: linux-2.6.24/fs/devpts/inode.c
>>> ===
>>> --- linux-2.6.24.orig/fs/devpts/inode.c 2008-02-05 17:30:52.000000000 -0800
>>> +++ linux-2.6.24/fs/devpts/inode.c 2008-02-05 19:16:39.000000000 -0800
>>> @@ -34,7 +34,10 @@ static inline struct idr *current_pts_ns
>>> }
>>>
>>> static struct vfsmount *devpts_mnt;
>>> -static struct dentry *devpts_root;
>>> +static inline struct vfsmount *current_pts_ns_mnt(void)
>>> +{
>>> + return devpts_mnt;
>>> +}
>>>
>>> static struct {
>>> int setuid;

```
>>> @@ -130,7 +133,7 @@ devpts_fill_super(struct super_block *s,
>>>   inode->i_fop = &simple_dir_operations;
>>>   inode->i_nlink = 2;
>>>
>>> - devpts_root = s->s_root = d_alloc_root(inode);
>>> + s->s_root = d_alloc_root(inode);
>>>   if (s->s_root)
>>>    return 0;
>>>
>>> @@ -140,10 +143,53 @@ fail:
>>>   return -ENOMEM;
>>>  }
>>>
>>> +/*
>>> + * We use test and set super-block operations to help determine whether we
>>> + * need a new super-block for this namespace. get_sb() walks the list of
>>> + * existing devpts supers, comparing them with the @data ptr. Since we
>>> + * passed 'current's namespace as the @data pointer we can compare the
>>> + * namespace pointer in the super-block's 's_fs_info'.  If the test is
>>> + * TRUE then get_sb() returns a new active reference to the super block.
>>> + * Otherwise, it helps us build an active reference to a new one.
>>> + */
>>> +
>>> +static int devpts_test_sb(struct super_block *sb, void *data)
>>> +{
>>> + return sb->s_fs_info == data;
>>> +}
>>> +
>>> +static int devpts_set_sb(struct super_block *sb, void *data)
>>> +{
>>> + sb->s_fs_info = data;
>>> + return set_anon_super(sb, NULL);
>>> +}
>>> +
>>>  static int devpts_get_sb(struct file_system_type *fs_type,
>>>   int flags, const char *dev_name, void *data, struct vfsmount *mnt)
>>>  {
>>> - return get_sb_single(fs_type, flags, data, devpts_fill_super, mnt);
>>> + struct super_block *sb;
>>> + int err;
>>> +
>>> + /* hereafter we're very simlar to get_sb_nodev */
>>> + sb = sget(fs_type, devpts_test_sb, devpts_set_sb, data);
>>> + if (IS_ERR(sb))
>>> +  return PTR_ERR(sb);
>>> +
>>> + if (sb->s_root)
>>> +  return simple_set_mnt(mnt, sb);
```

>>> +
>>> + sb->s_flags = flags;
>>> + err = devpts_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
>>> + if (err) {
>>> + up_write(&sb->s_umount);
>>> + deactivate_super(sb);
>>> + return err;
>>> + }
>>> +
>> That stuff becomes very very similar to that in proc :)
>> Makes sense to consolidate. Maybe...
>
> Yeah, and the mqns that Cedric sent too. I think Cedric said he'd
> started an a patch implementing a helper. Cedric?

yes.

it's basically a get_sb_single_per_ns() routine using ->s_fs_info
to distinguish the ns but there seems to be more to do to support
correctly namespaces using internal filesystems (circular ref)

C.

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Oren Laadan on Wed, 06 Feb 2008 19:25:09 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> Serge E. Hallyn wrote:
>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>> sukadev@us.ibm.com wrote:
>>>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>>>> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
[SNIP]

>>>> That stuff becomes very very similar to that in proc :)
>>>> Makes sense to consolidate. Maybe...
>>> Yeah, and the mqns that Cedric sent too. I think Cedric said he'd
>>> started an a patch implementing a helper. Cedric?
>> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
>> but the thread was abandoned by the time I decided to do-it-right-now.

>>
>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
>> this makes impossible to push a bind mount inside a new namespace, which
>> may operate in some chroot environment. But this ability is heavily
>
> Which direction do you want to go? I'm wondering whether mounts
> propagation can address it.
>
> Though really, I think you're right - we shouldn't break the kernel
> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
> force the combination.
>
>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
>> that would be very very good :) See my next comment about this issue.
>>
>>> Pavel, not long ago you said you were starting to look at tty and pty
>>> stuff - did you have any different ideas on devpts virtualization, or
>>> are you ok with this minus your comments thus far?
>> I have a similar idea of how to implement this, but I didn't thought
>> about the details. As far as this issue is concerned, I see no reasons
>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
>> we may safely hold the ptsns from the superblock and be happy. The
>> same seems applicable to the mqns, no?
>
> But the current->nsproxy->devpts->mnt is used in several functions in
> patch 3.
>
>> The reason I have the kern_mount-ed instance of proc for pid namespaces
>> is that I need a vfsmount to flush task entries from, but allowing
>> it to be NULL (i.e. no kern_mount, but optional user mounts) means
>> handing all the possible races, which is too heavy. But do we actually
>> need the vfsmount for devpts and mqns if no user-space mounts exist?
>>
>> Besides, I planned to include legacy ptys virtualization and console
>> virtualizatin in this namespace, but it seems, that it is not present
>> in this particular one.
>
> I had been thinking the consoles would have their own ns, since there's
> really nothing linking them, but there really is no good reason why
> userspace should ever want them separate. So I'm fine with combining
> them.

If you want to run something like an X server inside each container
(eg each container holds a desktop session of a different user), then
you need a separate virtual-console namespace for each container.

(yes, X per-se needs to provide remote display as opposed to use
local hardware; see http://www.ncl.cs.columbia.edu/research/thinc/)

[SNIP]

Oren.

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by serue on Wed, 06 Feb 2008 19:37:24 GMT
View Forum Message <> Reply to Message

Quoting Oren Laadan (orenl@cs.columbia.edu):
>
>
> Serge E. Hallyn wrote:
>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>> Serge E. Hallyn wrote:
>>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>>> sukadev@us.ibm.com wrote:
>>>>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>>>>> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
> [SNIP]
>
>>>>> That stuff becomes very very similar to that in proc :)
>>>>> Makes sense to consolidate. Maybe...
>>>> Yeah, and the mqns that Cedric sent too. I think Cedric said he'd
>>>> started an a patch implementing a helper. Cedric?
>>> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
>>> but the thread was abandoned by the time I decided to do-it-right-now.
>>>
>>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
>>> this makes impossible to push a bind mount inside a new namespace, which
>>> may operate in some chroot environment. But this ability is heavily
>> Which direction do you want to go? I'm wondering whether mounts
>> propagation can address it.
>> Though really, I think you're right - we shouldn't break the kernel
>> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
>> force the combination.
>>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
>>> that would be very very good :) See my next comment about this issue.
>>>
>>>> Pavel, not long ago you said you were starting to look at tty and pty
>>>> stuff - did you have any different ideas on devpts virtualization, or
>>>> are you ok with this minus your comments thus far?

>>> I have a similar idea of how to implement this, but I didn't thought
>>> about the details. As far as this issue is concerned, I see no reasons
>>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
>>> we may safely hold the ptsns from the superblock and be happy. The
>>> same seems applicable to the mqns, no?
>> But the current->nsproxy->devpts->mnt is used in several functions in
>> patch 3.
>>> The reason I have the kern_mount-ed instance of proc for pid namespaces
>>> is that I need a vfsmount to flush task entries from, but allowing
>>> it to be NULL (i.e. no kern_mount, but optional user mounts) means
>>> handing all the possible races, which is too heavy. But do we actually
>>> need the vfsmount for devpts and mqns if no user-space mounts exist?
>>>
>>> Besides, I planned to include legacy ptys virtualization and console
>>> virtualizatin in this namespace, but it seems, that it is not present
>>> in this particular one.
>> I had been thinking the consoles would have their own ns, since there's
>> really nothing linking them, but there really is no good reason why
>> userspace should ever want them separate. So I'm fine with combining
>> them.
>
> If you want to run something like an X server inside each container
> (eg each container holds a desktop session of a different user), then
> you need a separate virtual-console namespace for each container.

Ok, but whether the consoles and devpts are unshared with the same
cloneflag or not isn't an issue, right?

> (yes, X per-se needs to provide remote display as opposed to use
> local hardware; see http://www.ncl.cs.columbia.edu/research/thinc/)

-serge

Subject: Re: [RFC][PATCH 4/4]: Enable cloning PTY namespaces
Posted by serue on Wed, 06 Feb 2008 19:45:02 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> >>>>>>
> >>>>>> +struct pts_namespace *new_pts_ns(void)
> >>>>>> +{
> >>>>>> + struct pts_namespace *ns;
> >>>>>> +

> >>>>>> + ns = kmalloc(sizeof(*ns), GFP_KERNEL);
> >>>>>> + if (!ns)
> >>>>>> + return ERR_PTR(-ENOMEM);
> >>>>>> +
> >>>>>> + ns->mnt = kern_mount_data(&devpts_fs_type, ns);
> >>>>> You create a circular references here - the namespace
> >>>>> holds the vfsmnt, the vfsmnt holds a superblock, a superblock
> >>>>> holds the namespace.
> >>>> Hmm, yeah, good point. That was probably in my original version last
> >>>> year, so my fault not Suka's. Suka, would it work to have the
> >>>> sb->s_info point to the namespace but not grab a reference, than have
> >>> If you don't then you may be in situation, when this devpts
> >>> is mounted from userspace and in case the namespace is dead
> >>> superblock will point to garbage... Superblock MUST hold the
> >>> namespace :)
> >> But when the ns is freed sb->s_info would be NULL. Surely the helpers
> >> can be made to handle that safely?
> >
> > Hm... How do we find the proper superblock? Have a reference on
> > it from the namespace? I'm afraid it will be easy to resolve the
> > locking issues here.
> >
> > I propose another scheme - we simply don't have ANY references
> > from namespace to superblock/vfsmount, but get the current
> > namespace in devpts_get_sb() and put in devpts_free_sb().
>
> I've choosen another path in mq_ns.
>
> I also don't take any refcount on superblock/vfsmount of the new mq_ns
> bc of the circular ref. I've considered that namespaces only apply to
> processes : the refcount of a namespace is incremented each time a new
> task is cloned and the namespace (in my case mq_ns) is released when
> the last tasks exists. But this becomes an issue with user mounts which
> survives task death. you end up having a user mount pointing to a bogus
> mq_ns.
>
> unless you require to have CLONE_NEWNS at the sametime.
>
> Now, this CLONE_NEWNS enforcement seems to be an issue with bind mount.
>
> ... jumping to the other thread :)

But once again, given that the mnt/sb is a view into a namespace bound
to a set of tasks, if all those tasks have exited, I see nothing wrong
with having sb->s_info being made NULL, so that a task in another
namespace attempting to access the exited namespace through a user mount
sees an empty directory.

So again I recommend that we should simply have sb->s_info point to
the namespace but without taking a reference, and have free_x_ns() set
x_ns->mnt->sb->s_info to NULL. (That'll take a barrier of some kind,
which we can maybe build into the common helper)

-serge

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Oren Laadan on Wed, 06 Feb 2008 19:45:10 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Oren Laadan (orenl@cs.columbia.edu):
>>
>> Serge E. Hallyn wrote:
>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>> Serge E. Hallyn wrote:
>>>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>>>> sukadev@us.ibm.com wrote:
>>>>>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>>>>>> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
>> [SNIP]
>>
>>>>>> That stuff becomes very very similar to that in proc :)
>>>>>> Makes sense to consolidate. Maybe...
>>>>> Yeah, and the mqns that Cedric sent too. I think Cedric said he'd
>>>>> started an a patch implementing a helper. Cedric?
>>>> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
>>>> but the thread was abandoned by the time I decided to do-it-right-now.
>>>>
>>>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
>>>> this makes impossible to push a bind mount inside a new namespace, which
>>>> may operate in some chroot environment. But this ability is heavily
>>> Which direction do you want to go? I'm wondering whether mounts
>>> propagation can address it.
>>> Though really, I think you're right - we shouldn't break the kernel
>>> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
>>> force the combination.
>>>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
>>>> that would be very very good :) See my next comment about this issue.
>>>>
>>>>> Pavel, not long ago you said you were starting to look at tty and pty
>>>>> stuff - did you have any different ideas on devpts virtualization, or

>>>>> are you ok with this minus your comments thus far?
>>>> I have a similar idea of how to implement this, but I didn't thought
>>>> about the details. As far as this issue is concerned, I see no reasons
>>>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
>>>> we may safely hold the ptsns from the superblock and be happy. The
>>>> same seems applicable to the mqns, no?
>>> But the current->nsproxy->devpts->mnt is used in several functions in
>>> patch 3.
>>>> The reason I have the kern_mount-ed instance of proc for pid namespaces
>>>> is that I need a vfsmount to flush task entries from, but allowing
>>>> it to be NULL (i.e. no kern_mount, but optional user mounts) means
>>>> handing all the possible races, which is too heavy. But do we actually
>>>> need the vfsmount for devpts and mqns if no user-space mounts exist?
>>>>
>>>> Besides, I planned to include legacy ptys virtualization and console
>>>> virtualizatin in this namespace, but it seems, that it is not present
>>>> in this particular one.
>>> I had been thinking the consoles would have their own ns, since there's
>>> really nothing linking them, but there really is no good reason why
>>> userspace should ever want them separate. So I'm fine with combining
>>> them.
>> If you want to run something like an X server inside each container
>> (eg each container holds a desktop session of a different user), then
>> you need a separate virtual-console namespace for each container.
>
> Ok, but whether the consoles and devpts are unshared with the same
> cloneflag or not isn't an issue, right?

true. (I misread your comment.)
(
modulo that we are additional-clone-flags-challenged ...)

>
>> (yes, X per-se needs to provide remote display as opposed to use
>> local hardware; see http://www.ncl.cs.columbia.edu/research/thinc/)
>
> -serge

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by serue on Wed, 06 Feb 2008 19:58:55 GMT
View Forum Message <> Reply to Message

Quoting Oren Laadan (orenl@cs.columbia.edu):

>
>
> Serge E. Hallyn wrote:
>> Quoting Oren Laadan (orenl@cs.columbia.edu):
>>>
>>> Serge E. Hallyn wrote:
>>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>>> Serge E. Hallyn wrote:
>>>>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>>>>> sukadev@us.ibm.com wrote:
>>>>>>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>>>>>>> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
>>> [SNIP]
>>>
>>>>>>> That stuff becomes very very similar to that in proc :)
>>>>>>> Makes sense to consolidate. Maybe...
>>>>>> Yeah, and the mqns that Cedric sent too. I think Cedric said he'd
>>>>>> started an a patch implementing a helper. Cedric?
>>>>> Mmm. I wanted to send one small objection to Cedric's patches with
>>>>> mqns,
>>>>> but the thread was abandoned by the time I decided to do-it-right-now.
>>>>>
>>>>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
>>>>> this makes impossible to push a bind mount inside a new namespace,
>>>>> which
>>>>> may operate in some chroot environment. But this ability is heavily
>>>> Which direction do you want to go? I'm wondering whether mounts
>>>> propagation can address it.
>>>> Though really, I think you're right - we shouldn't break the kernel
>>>> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
>>>> force the combination.
>>>>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
>>>>> that would be very very good :) See my next comment about this issue.
>>>>>
>>>>>> Pavel, not long ago you said you were starting to look at tty and pty
>>>>>> stuff - did you have any different ideas on devpts virtualization, or
>>>>>> are you ok with this minus your comments thus far?
>>>>> I have a similar idea of how to implement this, but I didn't thought
>>>>> about the details. As far as this issue is concerned, I see no reasons
>>>>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
>>>>> we may safely hold the ptsns from the superblock and be happy. The
>>>>> same seems applicable to the mqns, no?
>>>> But the current->nsproxy->devpts->mnt is used in several functions in
>>>> patch 3.
>>>>> The reason I have the kern_mount-ed instance of proc for pid namespaces
>>>>> is that I need a vfsmount to flush task entries from, but allowing
>>>>> it to be NULL (i.e. no kern_mount, but optional user mounts) means
>>>>> handing all the possible races, which is too heavy. But do we actually

>>>>> need the vfsmount for devpts and mqns if no user-space mounts exist?
>>>>>
>>>>> Besides, I planned to include legacy ptys virtualization and console
>>>>> virtualizatin in this namespace, but it seems, that it is not present
>>>>> in this particular one.
>>>> I had been thinking the consoles would have their own ns, since there's
>>>> really nothing linking them, but there really is no good reason why
>>>> userspace should ever want them separate. So I'm fine with combining
>>>> them.
>>> If you want to run something like an X server inside each container
>>> (eg each container holds a desktop session of a different user), then
>>> you need a separate virtual-console namespace for each container.
>> Ok, but whether the consoles and devpts are unshared with the same
>> cloneflag or not isn't an issue, right?
>
> true. (I misread your comment.)
> (
> modulo that we are additional-clone-flags-challenged ...)

Right, plus the fact that the number of clone flags involved becomes
almost obscene. Let's see if Pavel and Suka have a preference, since
one of them seems likely to end up coding it :)

>>> (yes, X per-se needs to provide remote display as opposed to use
>>> local hardware; see http://www.ncl.cs.columbia.edu/research/thinc/)

Nice, by the way :)

>> -serge

thanks,
-serge

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Pavel Emelianov on Thu, 07 Feb 2008 09:43:36 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Pavel Emelyanov (xemul@openvz.org):
>> [snip]
>>
>>>> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
>>>> but the thread was abandoned by the time I decided to do-it-right-now.

>>>>
>>>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
>>>> this makes impossible to push a bind mount inside a new namespace, which
>>>> may operate in some chroot environment. But this ability is heavily
>>> Which direction do you want to go? I'm wondering whether mounts
>>> propagation can address it.
>> Hardly. AFAIS there's no way to let the chroot-ed tasks see parts of
>> vfs tree, that left behind them after chroot, unless they are in the
>> same mntns as you, and you bind mount this parts to their tree. No?
>
> Well no, but I suspect I'm just not understanding what you want to do.
> But if the chroot is under /jail1, and you've done, say,
>
> mkdir -p /share/pts
> mkdir -p /jail1/share
> mount --bind /share /share
> mount --make-shared /share
> mount --bind /share /jail1/share
> mount --make-slave /jail1/share
>
> before the chroot-ed tasks were cloned with CLONE_NEWNS, then when you
> do
>
> mount --bind /dev/pts /share/pts
>
> from the parent mntns (not that I know why you'd want to do *that* :)
> then the chroot'ed tasks will see the original mntns's /dev/pts under
> /jail1/share.

I haven't yet tried that, but :(this function

```
 static inline int check_mnt(struct vfsmount *mnt)
 {
        return mnt->mnt_ns == current->nsproxy->mnt_ns;
 }
```

and this code in do_loopback

```
        if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
                goto out;
```

makes me think that trying to bind a mount from another mntns
ot _to_ another is prohibited... Do I miss something?

>>> Though really, I think you're right - we shouldn't break the kernel
>>> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
>>> force the combination.
>>>

>>>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
>>>> that would be very very good :) See my next comment about this issue.
>>>>
>>>>> Pavel, not long ago you said you were starting to look at tty and pty
>>>>> stuff - did you have any different ideas on devpts virtualization, or
>>>>> are you ok with this minus your comments thus far?
>>>> I have a similar idea of how to implement this, but I didn't thought
>>>> about the details. As far as this issue is concerned, I see no reasons
>>>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
>>>> we may safely hold the ptsns from the superblock and be happy. The
>>>> same seems applicable to the mqns, no?
>>> But the current->nsproxy->devpts->mnt is used in several functions in
>>> patch 3.
>> Indeed. I overlooked this. Then we're in a deep ... problem here.
>>
>> Breaking this circle was not that easy with pid namespaces, so
>> I put the strut in proc_flush_task - when the last task from the
>> namespace exits the kern-mount-ed vfsmnt is dropped, but we can't
>> do the same stuff with devpts.
>
> But I still don't see what the problem is with my proposal? So long as
> you agree that if there are no tasks remaining in the devptsns,
> then any task which has its devpts mounted should see an empty directory
> (due to sb->s_info being NULL), I think it works.

Well, if we _do_ can handle the races with ns->devpts_mnt switch
from not NULL to NULL, then I'm fine with this approach.

I just remember, that with pid namespaces this caused a complicated
locking and performance degradation. This is the problem I couldn't
remember yesterday.

>> I do not remember now what the problem was and it's already quite
>> late in Moscow, so if you don't mind I'll revisit the issue tomorrow.
>
> Ok, that's fine. I'll let it sit until then too :) Good night.
>
>> Off-topic: does any of you know whether Andrew is willing to accept
>> new features in the nearest future? The problem is that I have a
>> device visibility controller fixed and pending to send, but I can't
>> guess a good time for it :)
>
> Well even if Andrew won't take it I'd like to see it, so I'd appreciate
> a resend.
>
>>>> The reason I have the kern_mount-ed instance of proc for pid namespaces
>>>> is that I need a vfsmount to flush task entries from, but allowing
>>>> it to be NULL (i.e. no kern_mount, but optional user mounts) means

>>>> handing all the possible races, which is too heavy. But do we actually
>>>> need the vfsmount for devpts and mqns if no user-space mounts exist?
>>>>
>>>> Besides, I planned to include legacy ptys virtualization and console
>>>> virtualizatin in this namespace, but it seems, that it is not present
>>>> in this particular one.
>>> I had been thinking the consoles would have their own ns, since there's
>>> really nothing linking them, but there really is no good reason why
>>> userspace should ever want them separate. So I'm fine with combining
>>> them.
>> OK.
>

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Cedric Le Goater on Thu, 07 Feb 2008 10:17:27 GMT
View Forum Message <> Reply to Message

>>> Breaking this circle was not that easy with pid namespaces, so
>>> I put the strut in proc_flush_task - when the last task from the
>>> namespace exits the kern-mount-ed vfsmnt is dropped, but we can't
>>> do the same stuff with devpts.
>> But I still don't see what the problem is with my proposal? So long as
>> you agree that if there are no tasks remaining in the devptsns,
>> then any task which has its devpts mounted should see an empty directory
>> (due to sb->s_info being NULL), I think it works.
>
> Well, if we _do_ can handle the races with ns->devpts_mnt switch
> from not NULL to NULL, then I'm fine with this approach.

I 'll take a look at it for the mq namespace.

we will need to flush the dcache in some way nop ? to make sure the lookup
in the directory fails to return anything after the ns has become NULL.
I'm not an fs expert so I might be completely wrong there but I'll study
in this direction to see if we can drop the CLONE_NEWNS.

> I just remember, that with pid namespaces this caused a complicated
> locking and performance degradation. This is the problem I couldn't
> remember yesterday.

That might have been bc you had to invalidate the /proc dentries ?

C.

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Cedric Le Goater on Thu, 07 Feb 2008 10:25:09 GMT
View Forum Message <> Reply to Message

>> Off-topic: does any of you know whether Andrew is willing to accept
>> new features in the nearest future? The problem is that I have a
>> device visibility controller fixed and pending to send, but I can't
>> guess a good time for it :)

I have the clone64/unshare64 syscalls ready for most common arches :

x86, x86_64, x86_64(32), ppc64, ppc64(32), s390x, s390x(31)

do you care to review or shall I send directly to andrew ?

There's a freezer patchset also that I need to resend ...

C.

> Well even if Andrew won't take it I'd like to see it, so I'd appreciate
> a resend.

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Pavel Emelianov on Thu, 07 Feb 2008 10:50:46 GMT
View Forum Message <> Reply to Message

Cedric Le Goater wrote:
>>> Off-topic: does any of you know whether Andrew is willing to accept
>>> new features in the nearest future? The problem is that I have a
>>> device visibility controller fixed and pending to send, but I can't
>>> guess a good time for it :)
>
> I have the clone64/unshare64 syscalls ready for most common arches :
>
> x86, x86_64, x86_64(32), ppc64, ppc64(32), s390x, s390x(31)

>
> do you care to review or shall I send directly to andrew ?

I think you can send them t Andrew :)

> There's a freezer patchset also that I need to resend ...
>
> C.
>
>> Well even if Andrew won't take it I'd like to see it, so I'd appreciate
>> a resend.
>

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by serue on Thu, 07 Feb 2008 14:22:35 GMT
View Forum Message <> Reply to Message

Quoting Pavel Emelyanov (xemul@openvz.org):
> Serge E. Hallyn wrote:
> > Quoting Pavel Emelyanov (xemul@openvz.org):
> >> [snip]
> >>
> >>>> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
> >>>> but the thread was abandoned by the time I decided to do-it-right-now.
> >>>>
> >>>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
> >>>> this makes impossible to push a bind mount inside a new namespace, which
> >>>> may operate in some chroot environment. But this ability is heavily
> >>> Which direction do you want to go? I'm wondering whether mounts
> >>> propagation can address it.
> >> Hardly. AFAIS there's no way to let the chroot-ed tasks see parts of
> >> vfs tree, that left behind them after chroot, unless they are in the
> >> same mntns as you, and you bind mount this parts to their tree. No?
> >
> > Well no, but I suspect I'm just not understanding what you want to do.
> > But if the chroot is under /jail1, and you've done, say,
> >
> > mkdir -p /share/pts
> > mkdir -p /jail1/share
> > mount --bind /share /share
> > mount --make-shared /share
> > mount --bind /share /jail1/share

> > mount --make-slave /jail1/share
> >
> > before the chroot-ed tasks were cloned with CLONE_NEWNS, then when you
> > do
> >
> > mount --bind /dev/pts /share/pts
> >
> > from the parent mntns (not that I know why you'd want to do *that* :)
> > then the chroot'ed tasks will see the original mntns's /dev/pts under
> > /jail1/share.
>
> I haven't yet tried that, but :(this function
>
> static inline int check_mnt(struct vfsmount *mnt)
> {
>   return mnt->mnt_ns == current->nsproxy->mnt_ns;
> }
>
> and this code in do_loopback
>
>   if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
>   goto out;
>
> makes me think that trying to bind a mount from another mntns
> ot _to_ another is prohibited... Do I miss something?

That's used at the top of explicit mounting paths, so if you found a way
to access a nameidata in the other mnt_ns and tried to mount /dev/pts
straight onto that nd this check would cause it to fail. But what I
described above mounts onto /share/pts, which is in the same ns. Then
the mouts propagation code in fs/pnode.c forwards the mount into the
other namespace.

Still I suspect I wasn't quite thinking right. If the target task had
already umounted /dev/pts and remounted it, there would be nothing to
forward your bind mount to and so nothing would happen.

Still that's moot :) Either we should find a way to get rid of the
CLONE_NEWNS requirement, or we should provide a cgroup to access
/dev/pts under the cgroup file tree.

> >>> Though really, I think you're right - we shouldn't break the kernel
> >>> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
> >>> force the combination.
> >>>
> >>>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
> >>>> that would be very very good :) See my next comment about this issue.
> >>>>

> >>>>> Pavel, not long ago you said you were starting to look at tty and pty
> >>>>> stuff - did you have any different ideas on devpts virtualization, or
> >>>>> are you ok with this minus your comments thus far?
> >>>> I have a similar idea of how to implement this, but I didn't thought
> >>>> about the details. As far as this issue is concerned, I see no reasons
> >>>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
> >>>> we may safely hold the ptsns from the superblock and be happy. The
> >>>> same seems applicable to the mqns, no?
> >>> But the current->nsproxy->devpts->mnt is used in several functions in
> >>> patch 3.
> >> Indeed. I overlooked this. Then we're in a deep ... problem here.
> >>
> >> Breaking this circle was not that easy with pid namespaces, so
> >> I put the strut in proc_flush_task - when the last task from the
> >> namespace exits the kern-mount-ed vfsmnt is dropped, but we can't
> >> do the same stuff with devpts.
> >
> > But I still don't see what the problem is with my proposal? So long as
> > you agree that if there are no tasks remaining in the devptsns,
> > then any task which has its devpts mounted should see an empty directory
> > (due to sb->s_info being NULL), I think it works.
>
> Well, if we _do_ can handle the races with ns->devpts_mnt switch
> from not NULL to NULL, then I'm fine with this approach.
>
> I just remember, that with pid namespaces this caused a complicated
> locking and performance degradation. This is the problem I couldn't
> remember yesterday.

Yeah it sure seems like there must be some gotcha in there somewhere...

> >> I do not remember now what the problem was and it's already quite
> >> late in Moscow, so if you don't mind I'll revisit the issue tomorrow.
> >
> > Ok, that's fine. I'll let it sit until then too :) Good night.
> >
> >> Off-topic: does any of you know whether Andrew is willing to accept
> >> new features in the nearest future? The problem is that I have a
> >> device visibility controller fixed and pending to send, but I can't
> >> guess a good time for it :)
> >
> > Well even if Andrew won't take it I'd like to see it, so I'd appreciate
> > a resend.
> >
> >>>> The reason I have the kern_mount-ed instance of proc for pid namespaces
> >>>> is that I need a vfsmount to flush task entries from, but allowing
> >>>> it to be NULL (i.e. no kern_mount, but optional user mounts) means
> >>>> handing all the possible races, which is too heavy. But do we actually

> >>>> need the vfsmount for devpts and mqns if no user-space mounts exist?
> >>>>
> >>>> Besides, I planned to include legacy ptys virtualization and console
> >>>> virtualizatin in this namespace, but it seems, that it is not present
> >>>> in this particular one.
> >>> I had been thinking the consoles would have their own ns, since there's
> >>> really nothing linking them, but there really is no good reason why
> >>> userspace should ever want them separate. So I'm fine with combining
> >>> them.
> >> OK.
> >

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Oren Laadan on Tue, 12 Feb 2008 00:34:47 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn wrote:
> Quoting Oren Laadan (orenl@cs.columbia.edu):
>>
>> Serge E. Hallyn wrote:
>>> Quoting Oren Laadan (orenl@cs.columbia.edu):
>>>> Serge E. Hallyn wrote:
>>>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>>>> Serge E. Hallyn wrote:
>>>>>>> Quoting Pavel Emelyanov (xemul@openvz.org):
>>>>>>>> sukadev@us.ibm.com wrote:
>>>>>>>>> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
>>>>>>>>> Subject: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
>>>> [SNIP]
>>>>

[SNIP again]

>>>>> Besides, I planned to include legacy ptys virtualization and console
>>>>> virtualizatin in this namespace, but it seems, that it is not present
>>>>> in this particular one.
>>>> I had been thinking the consoles would have their own ns, since there's
>>>> really nothing linking them, but there really is no good reason why
>>>> userspace should ever want them separate. So I'm fine with combining
>>>> them.
>> If you want to run something like an X server inside each container
>> (eg each container holds a desktop session of a different user), then
>> you need a separate virtual-console namespace for each container.

>>> Ok, but whether the consoles and devpts are unshared with the same
>>> cloneflag or not isn't an issue, right?
>> true. (I misread your comment.)
>> (
>> modulo that we are additional-clone-flags-challenged ...)
>
> Right, plus the fact that the number of clone flags involved becomes
> almost obscene. Let's see if Pavel and Suka have a preference, since
> one of them seems likely to end up coding it :)
>
>>>> (yes, X per-se needs to provide remote display as opposed to use
>>>> local hardware; see http://www.ncl.cs.columbia.edu/research/thinc/)
>
> Nice, by the way :)
>

Thanks :)

Still off-topic, this is even nicer (also requires ultrafast checkpoint):
http://www.ncl.cs.columbia.edu/publications/sosp2007_dejaview

>>> -serge
>
> thanks,
> -serge

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Sukadev Bhattiprolu on Thu, 14 Feb 2008 18:16:58 GMT
View Forum Message <> Reply to Message

Serge E. Hallyn [serue@us.ibm.com] wrote:
|
| > exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
| > that would be very very good :) See my next comment about this issue.
| >
| > > Pavel, not long ago you said you were starting to look at tty and pty
| > > stuff - did you have any different ideas on devpts virtualization, or
| > > are you ok with this minus your comments thus far?
| >
| > I have a similar idea of how to implement this, but I didn't thought
| > about the details. As far as this issue is concerned, I see no reasons
| > why we need a kern_mount-ed devtpsfs instance. If we don't make such,
| > we may safely hold the ptsns from the superblock and be happy. The

| > same seems applicable to the mqns, no?
|
| But the current->nsproxy->devpts->mnt is used in several functions in
| patch 3.

Hmm, current_pts_ns_mnt() is used in:

 devpts_pty_new()
 devpts_get_tty()
 devpts_pty_kill()

All of these return error if current_pts_ns_mnt() returns NULL.
So, can we require user-space mount and unmount /dev/pts and return
error if any operation is attempted before the mount ?

|
| > The reason I have the kern_mount-ed instance of proc for pid namespaces
| > is that I need a vfsmount to flush task entries from, but allowing
| > it to be NULL (i.e. no kern_mount, but optional user mounts) means
| > handing all the possible races, which is too heavy. But do we actually
| > need the vfsmount for devpts and mqns if no user-space mounts exist?
| >
| > Besides, I planned to include legacy ptys virtualization and console
| > virtualizatin in this namespace, but it seems, that it is not present
| > in this particular one.
|
| I had been thinking the consoles would have their own ns, since there's
| really nothing linking them, but there really is no good reason why
| userspace should ever want them separate. So I'm fine with combining
| them.
|
| > >>> + sb->s_flags |= MS_ACTIVE;
| > >>> + devpts_mnt = mnt;
| > >>> +
| > >>> + return simple_set_mnt(mnt, sb);
| > >>> }
| > >>>
| > >>> static struct file_system_type devpts_fs_type = {
| > >>> @@ -158,10 +204,9 @@ static struct file_system_type devpts_fs
| > >>>   * to the System V naming convention
| > >>>   */
| > >>>
| > >>> -static struct dentry *get_node(int num)
| > >>> +static struct dentry *get_node(struct dentry *root, int num)
| > >>> {
| > >>>   char s[12];
| > >>> - struct dentry *root = devpts_root;
| > >>>   mutex_lock(&root->d_inode->i_mutex);

```
| > >>>   return lookup_one_len(s, root, sprintf(s, "%d", num));
| > >>>  }
| > >>> @@ -207,12 +252,28 @@ int devpts_pty_new(struct tty_struct *tt
| > >>>   struct tty_driver *driver = tty->driver;
| > >>>   dev_t device = MKDEV(driver->major, driver->minor_start+number);
| > >>>   struct dentry *dentry;
| > >>> - struct inode *inode = new_inode(devpts_mnt->mnt_sb);
| > >>> + struct dentry *root;
| > >>> + struct vfsmount *mnt;
| > >>> + struct inode *inode;
| > >>> +
| > >>>
| > >>>   /* We're supposed to be given the slave end of a pty */
| > >>>   BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
| > >>>   BUG_ON(driver->subtype != PTY_TYPE_SLAVE);
| > >>>
| > >>> + mnt = current_pts_ns_mnt();
| > >>> + if (!mnt)
| > >>> +  return -ENOSYS;
| > >>> + root = mnt->mnt_root;
| > >>> +
| > >>> + mutex_lock(&root->d_inode->i_mutex);
| > >>> + inode = idr_find(current_pts_ns_allocated_ptys(), number);
| > >>> + mutex_unlock(&root->d_inode->i_mutex);
| > >>> +
| > >>> + if (inode && !IS_ERR(inode))
| > >>> +  return -EEXIST;
| > >>> +
| > >>> + inode = new_inode(mnt->mnt_sb);
| > >>>   if (!inode)
| > >>>    return -ENOMEM;
| > >>>
| > >>> @@ -222,23 +283,31 @@ int devpts_pty_new(struct tty_struct *tt
| > >>>   inode->i_mtime = inode->i_atime = inode->i_ctime = CURRENT_TIME;
| > >>>   init_special_inode(inode, S_IFCHR|config.mode, device);
| > >>>   inode->i_private = tty;
| > >>> + idr_replace(current_pts_ns_allocated_ptys(), inode, number);
| > >>>
| > >>> - dentry = get_node(number);
| > >>> + dentry = get_node(root, number);
| > >>>   if (!IS_ERR(dentry) && !dentry->d_inode) {
| > >>>    d_instantiate(dentry, inode);
| > >>> -  fsnotify_create(devpts_root->d_inode, dentry);
| > >>> +  fsnotify_create(root->d_inode, dentry);
| > >>>   }
| > >>>
| > >>> - mutex_unlock(&devpts_root->d_inode->i_mutex);
| > >>> + mutex_unlock(&root->d_inode->i_mutex);
```

```
| > >>>
| > >>>   return 0;
| > >>>  }
| > >>>
| > >>>  struct tty_struct *devpts_get_tty(int number)
| > >>>  {
| > >>> - struct dentry *dentry = get_node(number);
| > >>> + struct vfsmount *mnt;
| > >>> + struct dentry *dentry;
| > >>>   struct tty_struct *tty;
| > >>>
| > >>> + mnt = current_pts_ns_mnt();
| > >>> + if (!mnt)
| > >>> +  return NULL;
| > >>> +
| > >>> + dentry = get_node(mnt->mnt_root, number);
| > >>> +
| > >>>   tty = NULL;
| > >>>   if (!IS_ERR(dentry)) {
| > >>>    if (dentry->d_inode)
| > >>> @@ -246,14 +315,21 @@ struct tty_struct *devpts_get_tty(int nu
| > >>>    dput(dentry);
| > >>>   }
| > >>>
| > >>> - mutex_unlock(&devpts_root->d_inode->i_mutex);
| > >>> + mutex_unlock(&mnt->mnt_root->d_inode->i_mutex);
| > >>>
| > >>>   return tty;
| > >>>  }
| > >>>
| > >>>  void devpts_pty_kill(int number)
| > >>>  {
| > >>> - struct dentry *dentry = get_node(number);
| > >>> + struct dentry *dentry;
| > >>> + struct dentry *root;
| > >>> + struct vfsmount *mnt;
| > >>> +
| > >>> + mnt = current_pts_ns_mnt();
| > >>> + root = mnt->mnt_root;
| > >>> +
| > >>> + dentry = get_node(root, number);
| > >>>
| > >>>   if (!IS_ERR(dentry)) {
| > >>>    struct inode *inode = dentry->d_inode;
| > >>> @@ -264,17 +340,23 @@ void devpts_pty_kill(int number)
| > >>>    }
| > >>>    dput(dentry);
| > >>>   }
```

| > >>> - mutex_unlock(&devpts_root->d_inode->i_mutex);
| > >>> + mutex_unlock(&root->d_inode->i_mutex);
| > >>> }
| > >>>
| > >>> static int __init init_devpts_fs(void)
| > >>> {
| > >>> - int err = register_filesystem(&devpts_fs_type);
| > >>> - if (!err) {
| > >>> - devpts_mnt = kern_mount(&devpts_fs_type);
| > >>> - if (IS_ERR(devpts_mnt))
| > >>> -   err = PTR_ERR(devpts_mnt);
| > >>> - }
| > >>> + struct vfsmount *mnt;
| > >>> + int err;
| > >>> +
| > >>> + err = register_filesystem(&devpts_fs_type);
| > >>> + if (err)
| > >>> + return err;
| > >>> +
| > >>> + mnt = kern_mount_data(&devpts_fs_type, NULL);
| > >>> + if (IS_ERR(mnt))
| > >>> + err = PTR_ERR(mnt);
| > >>> + else
| > >>> + devpts_mnt = mnt;
| > >>>   return err;
| > >>> }
| > >>>
| > >>> _____
| > >>> 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
| > >

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

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Sukadev Bhattiprolu on Thu, 14 Feb 2008 23:50:27 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| Serge E. Hallyn wrote:
| > Quoting Pavel Emelyanov (xemul@openvz.org):
| >> [snip]
| >>
| >>>> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
| >>>> but the thread was abandoned by the time I decided to do-it-right-now.
| >>>>
| >>>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
| >>>> this makes impossible to push a bind mount inside a new namespace, which
| >>>> may operate in some chroot environment. But this ability is heavily
| >>> Which direction do you want to go? I'm wondering whether mounts
| >>> propagation can address it.
| >> Hardly. AFAIS there's no way to let the chroot-ed tasks see parts of
| >> vfs tree, that left behind them after chroot, unless they are in the
| >> same mntns as you, and you bind mount this parts to their tree. No?
| >
| > Well no, but I suspect I'm just not understanding what you want to do.
| > But if the chroot is under /jail1, and you've done, say,
| >
| > mkdir -p /share/pts
| > mkdir -p /jail1/share
| > mount --bind /share /share
| > mount --make-shared /share
| > mount --bind /share /jail1/share
| > mount --make-slave /jail1/share
| >
| > before the chroot-ed tasks were cloned with CLONE_NEWNS, then when you
| > do
| >
| > mount --bind /dev/pts /share/pts
| >
| > from the parent mntns (not that I know why you'd want to do *that* :)
| > then the chroot'ed tasks will see the original mntns's /dev/pts under
| > /jail1/share.
|
| I haven't yet tried that, but :(this function
|
| static inline int check_mnt(struct vfsmount *mnt)
| {
| return mnt->mnt_ns == current->nsproxy->mnt_ns;
| }
|
| and this code in do_loopback
|

| 	if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
| 		goto out;
|
| makes me think that trying to bind a mount from another mntns
| ot _to_ another is prohibited... Do I miss something?
|
| >>> Though really, I think you're right - we shouldn't break the kernel
| >>> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
| >>> force the combination.
| >>>
| >>>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
| >>>> that would be very very good :) See my next comment about this issue.
| >>>>
| >>>>> Pavel, not long ago you said you were starting to look at tty and pty
| >>>>> stuff - did you have any different ideas on devpts virtualization, or
| >>>>> are you ok with this minus your comments thus far?
| >>>> I have a similar idea of how to implement this, but I didn't thought
| >>>> about the details. As far as this issue is concerned, I see no reasons
| >>>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
| >>>> we may safely hold the ptsns from the superblock and be happy. The
| >>>> same seems applicable to the mqns, no?
| >>> But the current->nsproxy->devpts->mnt is used in several functions in
| >>> patch 3.
| >> Indeed. I overlooked this. Then we're in a deep ... problem here.
| >>
| >> Breaking this circle was not that easy with pid namespaces, so
| >> I put the strut in proc_flush_task - when the last task from the
| >> namespace exits the kern-mount-ed vfsmnt is dropped, but we can't
| >> do the same stuff with devpts.
| >
| > But I still don't see what the problem is with my proposal? So long as
| > you agree that if there are no tasks remaining in the devptsns,
| > then any task which has its devpts mounted should see an empty directory
| > (due to sb->s_info being NULL), I think it works.
|
| Well, if we _do_ can handle the races with ns->devpts_mnt switch
| from not NULL to NULL, then I'm fine with this approach.
|
| I just remember, that with pid namespaces this caused a complicated
| locking and performance degradation. This is the problem I couldn't
| remember yesterday.

Well, iirc, one problem with pid namespaces was that we need to keep
the task and pid_namespace association until the task was waited on
(for instance the wait() call from parent needs the pid_t of the
child which is tied to the pid ns in struct upid).

For this reason, we don't drop the mnt reference in free_pid_ns() but

hold the reference till proc_flush_task().

With devpts, can't we simply drop the reference in free_pts_ns() so
that when the last task using the pts_ns exits, we can unmount and
release the mnt ?

IOW, do you suspect that the circular reference leads to leaking vfsmnts ?

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

---

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Pavel Emelianov on Fri, 15 Feb 2008 07:57:38 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Pavel Emelianov [xemul@openvz.org] wrote:
> | Serge E. Hallyn wrote:
> | > Quoting Pavel Emelyanov (xemul@openvz.org):
> | >> [snip]
> | >>
> | >>>> Mmm. I wanted to send one small objection to Cedric's patches with mqns,
> | >>>> but the thread was abandoned by the time I decided to do-it-right-now.
> | >>>>
> | >>>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
> | >>>> this makes impossible to push a bind mount inside a new namespace, which
> | >>>> may operate in some chroot environment. But this ability is heavily
> | >>> Which direction do you want to go? I'm wondering whether mounts
> | >>> propagation can address it.
> | >> Hardly. AFAIS there's no way to let the chroot-ed tasks see parts of
> | >> vfs tree, that left behind them after chroot, unless they are in the
> | >> same mntns as you, and you bind mount this parts to their tree. No?
> | >
> | > Well no, but I suspect I'm just not understanding what you want to do.
> | > But if the chroot is under /jail1, and you've done, say,
> | >
> | > mkdir -p /share/pts
> | > mkdir -p /jail1/share
> | > mount --bind /share /share
> | > mount --make-shared /share
> | > mount --bind /share /jail1/share
> | > mount --make-slave /jail1/share
> | >
> | > before the chroot-ed tasks were cloned with CLONE_NEWNS, then when you
> | > do
> | >

> | > mount --bind /dev/pts /share/pts
> | >
> | > from the parent mntns (not that I know why you'd want to do *that* :)
> | > then the chroot'ed tasks will see the original mntns's /dev/pts under
> | > /jail1/share.
> |
> | I haven't yet tried that, but :(this function
> |
> | static inline int check_mnt(struct vfsmount *mnt)
> | {
> | return mnt->mnt_ns == current->nsproxy->mnt_ns;
> | }
> |
> | and this code in do_loopback
> |
> | if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
> | goto out;
> |
> | makes me think that trying to bind a mount from another mntns
> | ot _to_ another is prohibited... Do I miss something?
> |
> | >>> Though really, I think you're right - we shouldn't break the kernel
> | >>> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
> | >>> force the combination.
> | >>>
> | >>>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
> | >>>> that would be very very good :) See my next comment about this issue.
> | >>>>
> | >>>>> Pavel, not long ago you said you were starting to look at tty and pty
> | >>>>> stuff - did you have any different ideas on devpts virtualization, or
> | >>>>> are you ok with this minus your comments thus far?
> | >>>> I have a similar idea of how to implement this, but I didn't thought
> | >>>> about the details. As far as this issue is concerned, I see no reasons
> | >>>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
> | >>>> we may safely hold the ptsns from the superblock and be happy. The
> | >>>> same seems applicable to the mqns, no?
> | >>> But the current->nsproxy->devpts->mnt is used in several functions in
> | >>> patch 3.
> | >> Indeed. I overlooked this. Then we're in a deep ... problem here.
> | >>
> | >> Breaking this circle was not that easy with pid namespaces, so
> | >> I put the strut in proc_flush_task - when the last task from the
> | >> namespace exits the kern-mount-ed vfsmnt is dropped, but we can't
> | >> do the same stuff with devpts.
> | >
> | > But I still don't see what the problem is with my proposal? So long as
> | > you agree that if there are no tasks remaining in the devptsns,
> | > then any task which has its devpts mounted should see an empty directory

> | > (due to sb->s_info being NULL), I think it works.
> |
> | Well, if we _do_ can handle the races with ns->devpts_mnt switch
> | from not NULL to NULL, then I'm fine with this approach.
> |
> | I just remember, that with pid namespaces this caused a complicated
> | locking and performance degradation. This is the problem I couldn't
> | remember yesterday.
>
> Well, iirc, one problem with pid namespaces was that we need to keep
> the task and pid_namespace association until the task was waited on
> (for instance the wait() call from parent needs the pid_t of the
> child which is tied to the pid ns in struct upid).
>
> For this reason, we don't drop the mnt reference in free_pid_ns() but
> hold the reference till proc_flush_task().
>
> With devpts, can't we simply drop the reference in free_pts_ns() so
> that when the last task using the pts_ns exits, we can unmount and
> release the mnt ?

I hope we can. The thing I'm worried about is whether we can correctly
handle race with this pointer switch from NULL to not-NULL.

> IOW, do you suspect that the circular reference leads to leaking vfsmnts ?
>

Of course! If the namespace holds the vfsmnt, vfsmnt holds the superblock
and the superblock holds the namespace we won't drop this chain ever,
unless some other object breaks this chain.

Subject: Re: [RFC][PATCH 3/4]: Enable multiple mounts of /dev/pts
Posted by Sukadev Bhattiprolu on Fri, 15 Feb 2008 17:52:07 GMT
View Forum Message <> Reply to Message

Pavel Emelianov [xemul@openvz.org] wrote:
| sukadev@us.ibm.com wrote:
| > Pavel Emelianov [xemul@openvz.org] wrote:
| > | Serge E. Hallyn wrote:
| > | > Quoting Pavel Emelyanov (xemul@openvz.org):
| > | >> [snip]
| > | >>
| > | >>>> Mmm. I wanted to send one small objection to Cedric's patches with mqns,

| > | >>>> but the thread was abandoned by the time I decided to do-it-right-now.
| > | >>>>
| > | >>>> So I can put it here: forcing the CLONE_NEWNS is not very good, since
| > | >>>> this makes impossible to push a bind mount inside a new namespace, which
| > | >>>> may operate in some chroot environment. But this ability is heavily
| > | >>> Which direction do you want to go? I'm wondering whether mounts
| > | >>> propagation can address it.
| > | >> Hardly. AFAIS there's no way to let the chroot-ed tasks see parts of
| > | >> vfs tree, that left behind them after chroot, unless they are in the
| > | >> same mntns as you, and you bind mount this parts to their tree. No?
| > | >
| > | > Well no, but I suspect I'm just not understanding what you want to do.
| > | > But if the chroot is under /jail1, and you've done, say,
| > | >
| > | > mkdir -p /share/pts
| > | > mkdir -p /jail1/share
| > | > mount --bind /share /share
| > | > mount --make-shared /share
| > | > mount --bind /share /jail1/share
| > | > mount --make-slave /jail1/share
| > | >
| > | > before the chroot-ed tasks were cloned with CLONE_NEWNS, then when you
| > | > do
| > | >
| > | > mount --bind /dev/pts /share/pts
| > | >
| > | > from the parent mntns (not that I know why you'd want to do *that* :)
| > | > then the chroot'ed tasks will see the original mntns's /dev/pts under
| > | > /jail1/share.
| > |
| > | I haven't yet tried that, but :(this function
| > |
| > | static inline int check_mnt(struct vfsmount *mnt)
| > | {
| > | return mnt->mnt_ns == current->nsproxy->mnt_ns;
| > | }
| > |
| > | and this code in do_loopback
| > |
| > | if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
| > | goto out;
| > |
| > | makes me think that trying to bind a mount from another mntns
| > | ot _to_ another is prohibited... Do I miss something?
| > |
| > | >>> Though really, I think you're right - we shouldn't break the kernel
| > | >>> doing CLONE_NEWMQ or CLONE_NEWPTS without CLONE_NEWNS, so we shouldn't
| > | >>> force the combination.

| > | >>>
| > | >>>> exploited in OpenVZ, so if we can somehow avoid forcing the NEWNS flag
| > | >>>> that would be very very good :) See my next comment about this issue.
| > | >>>>
| > | >>>>> Pavel, not long ago you said you were starting to look at tty and pty
| > | >>>>> stuff - did you have any different ideas on devpts virtualization, or
| > | >>>>> are you ok with this minus your comments thus far?
| > | >>>> I have a similar idea of how to implement this, but I didn't thought
| > | >>>> about the details. As far as this issue is concerned, I see no reasons
| > | >>>> why we need a kern_mount-ed devtpsfs instance. If we don't make such,
| > | >>>> we may safely hold the ptsns from the superblock and be happy. The
| > | >>>> same seems applicable to the mqns, no?
| > | >>> But the current->nsproxy->devpts->mnt is used in several functions in
| > | >>> patch 3.
| > | >> Indeed. I overlooked this. Then we're in a deep ... problem here.
| > | >>
| > | >> Breaking this circle was not that easy with pid namespaces, so
| > | >> I put the strut in proc_flush_task - when the last task from the
| > | >> namespace exits the kern-mount-ed vfsmnt is dropped, but we can't
| > | >> do the same stuff with devpts.
| > | >
| > | > But I still don't see what the problem is with my proposal? So long as
| > | > you agree that if there are no tasks remaining in the devptsns,
| > | > then any task which has its devpts mounted should see an empty directory
| > | > (due to sb->s_info being NULL), I think it works.
| > |
| > | Well, if we _do_ can handle the races with ns->devpts_mnt switch
| > | from not NULL to NULL, then I'm fine with this approach.
| > |
| > | I just remember, that with pid namespaces this caused a complicated
| > | locking and performance degradation. This is the problem I couldn't
| > | remember yesterday.
| >
| > Well, iirc, one problem with pid namespaces was that we need to keep
| > the task and pid_namespace association until the task was waited on
| > (for instance the wait() call from parent needs the pid_t of the
| > child which is tied to the pid ns in struct upid).
| >
| > For this reason, we don't drop the mnt reference in free_pid_ns() but
| > hold the reference till proc_flush_task().
| >
| > With devpts, can't we simply drop the reference in free_pts_ns() so
| > that when the last task using the pts_ns exits, we can unmount and
| > release the mnt ?
|
| I hope we can. The thing I'm worried about is whether we can correctly
| handle race with this pointer switch from NULL to not-NULL.
|

| > IOW, do you suspect that the circular reference leads to leaking vfsmnts ?
| >
|
| Of course! If the namespace holds the vfsmnt, vfsmnt holds the superblock
| and the superblock holds the namespace we won't drop this chain ever,
| unless some other object breaks this chain.

Of course :-) I had a bug in new_pts_ns() that was masking the problem.
I had

```
 ns->mnt  = kern_mount_data()...

 ...
 kref_init(&ns->kref);
```

So the kref_init() would overwrite the reference got by devpts_set_sb()
and was preventing the leaking vfsmnt in my test.

Thanks Pavel,

Sukadev

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