

## 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 1/7]: Propagate error code from devpts\_pty\_new

[PATCH 2/7]: Factor out PTY index allocation

[PATCH 3/7]: Enable multiple mounts of /dev/pts

[PATCH 4/7]: Implement get\_pts\_ns() and put\_pts\_ns()

[PATCH 5/7]: Determine pts\_ns from a pty's inode.

[PATCH 6/7]: Check for user-space mount of /dev/pts

[PATCH 7/7]: Enable cloning PTY namespaces

### Todo:

- This patchset depends on availability of additional clone flags. and relies on on Cedric's clone64 patchset.
- Needs more testing.

### Changelog[v1]:

- Fixed circular reference by not caching the pts\_ns in sb->s\_fs\_info (without incrementing reference count) and clearing the sb->s\_fs\_info when destroying the pts\_ns (See patch 3/7 for details).
- To allow access to a child container's ptys from parent container, determine the 'pts\_ns' of a 'pty' from its inode (See patch 5/7 for details).
- Added a check (hack) to ensure user-space mount of /dev/pts is done before creating PTYs in a new pts-ns (see patch 6/7 for details).
- Reorganized the patchset and removed redundant changes.
- Ported to work with Cedric Le Goater's clone64() system call now that we are out of clone\_flags.

### Changelog[v0]:

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: [PATCH 1/7] Propagate error code from devpts\_pty\_new  
Posted by [Sukadev Bhattiprolu](#) on Tue, 25 Mar 2008 04:22:13 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
Subject: [PATCH 1/7]: Propagate error code from devpts\_pty\_new

Have ptmx\_open() propagate any error code returned by devpts\_pty\_new().

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
Cc: Cedric Le Goater <clg@fr.ibm.com>  
Cc: Dave Hansen <haveblue@us.ibm.com>  
Cc: Serge Hallyn <serue@us.ibm.com>  
Cc: containers@lists.osdl.org

---

drivers/char/tty\_io.c | 4 ++--  
1 file changed, 2 insertions(+), 2 deletions(-)

Index: 2.6.25-rc5-mm1/drivers/char/tty\_io.c

```
=====
--- 2.6.25-rc5-mm1.orig/drivers/char/tty_io.c 2008-03-21 20:13:38.000000000 -0700
+++ 2.6.25-rc5-mm1/drivers/char/tty_io.c 2008-03-24 20:04:07.000000000 -0700
@@ -2835,8 +2835,8 @@ static int ptmx_open(struct inode *inode
     filp->private_data = tty;
     file_move(filp, &tty->tty_files);

-    retval = -ENOMEM;
-    if (devpts_pty_new(tty->link))
+    retval = devpts_pty_new(tty->link);
+    if (retval)
         goto out1;

     check_tty_count(tty, "tty_open");
```

---

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

---

---

Subject: [PATCH 2/7]: Factor out PTY index allocation  
Posted by [Sukadev Bhattiprolu](#) on Tue, 25 Mar 2008 04:23:42 GMT

---

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
Subject: [PATCH 2/7]: 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.

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

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

Signed-off-by: Matt Helsley<matthltc@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: 2.6.25-rc5-mm1/include/linux/devpts\_fs.h

```
=====
```

--- 2.6.25-rc5-mm1.orig/include/linux/devpts\_fs.h 2008-03-24 20:04:07.000000000 -0700  
+++ 2.6.25-rc5-mm1/include/linux/devpts\_fs.h 2008-03-24 20:04:26.000000000 -0700  
@@ -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) { }
```

Index: 2.6.25-rc5-mm1/drivers/char/tty\_io.c

```
=====
```

--- 2.6.25-rc5-mm1.orig/drivers/char/tty\_io.c 2008-03-24 20:04:07.000000000 -0700  
+++ 2.6.25-rc5-mm1/drivers/char/tty\_io.c 2008-03-24 20:04:26.000000000 -0700  
@@ -91,7 +91,6 @@  
#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>
@@ -137,9 +136,6 @@ 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 DEFINE_MUTEX(allocated_ptys_lock);
static int ptmx_open(struct inode *, struct file *);
#endif

@@ -2636,15 +2632,9 @@ static void release_dev(struct file *fil
 */
release_tty(tty, idx);

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

/**
@@ -2800,29 +2790,13 @@ static int ptmx_open(struct inode *inode
struct tty_struct *tty;
int retval;
int index;
- int idr_ret;

nonseekable_open(inode, filp);

/* find a device that is not in use. */
- mutex_lock(&allocated_ptys_lock);
- if (!idr_pre_get(&allocated_ptys, GFP_KERNEL)) {
- mutex_unlock(&allocated_ptys_lock);
- return -ENOMEM;
- }
- idr_ret = idr_get_new(&allocated_ptys, NULL, &index);
- if (idr_ret < 0) {

```

```

- mutex_unlock(&allocated_ptys_lock);
- if (idr_ret == -EAGAIN)
- return -ENOMEM;
- return -EIO;
- }
- if (index >= pty_limit) {
- idr_remove(&allocated_ptys, index);
- mutex_unlock(&allocated_ptys_lock);
- return -EIO;
- }
- mutex_unlock(&allocated_ptys_lock);
+ index = devpts_new_index();
+ if (index < 0)
+ return index;

```

```

    mutex_lock(&tty_mutex);
    retval = init_dev(ptm_driver, index, &tty);
@@ -2847,9 +2821,7 @@ out1:
    release_dev(filp);
    return retval;
out:
- mutex_lock(&allocated_ptys_lock);
- idr_remove(&allocated_ptys, index);
- mutex_unlock(&allocated_ptys_lock);
+ devpts_kill_index(index);
    return retval;
}
#endif

```

Index: 2.6.25-rc5-mm1/fs/devpts/inode.c

```

=====
--- 2.6.25-rc5-mm1.orig/fs/devpts/inode.c 2008-03-24 20:04:07.000000000 -0700
+++ 2.6.25-rc5-mm1/fs/devpts/inode.c 2008-03-24 20:04:26.000000000 -0700
@@ -17,6 +17,7 @@
#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>
@@ -26,6 +27,10 @@

#define DEVPTS_DEFAULT_MODE 0600

+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;
```

```
@@ -171,9 +176,44 @@ 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;

---

Containers mailing list

Containers@lists.linux-foundation.org

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

---

---

Subject: [PATCH 3/7]: Enable multiple mounts of /dev/pts

Posted by [Sukadev Bhattiprolu](#) on Tue, 25 Mar 2008 04:24:37 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

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

Subject: [PATCH 3/7]: 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\_mnt', 'devpts\_root' and uses a skeletal 'init\_pts\_ns' to store the vfsmount.

Changelog [v2]:

- (Pavel Emelianov/Serge Hallyn) Remove reference to pts\_ns from sb->s\_fs\_info to fix the circular reference (/dev/pts is not unmounted unless the pts\_ns is destroyed, so we don't need a reference to the pts\_ns).

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

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

Signed-off-by: Matt Helsley <matthltc@us.ibm.com>

---

fs/devpts/inode.c | 160 ++++++  
include/linux/devpts\_fs.h | 11 +++  
2 files changed, 143 insertions(+), 28 deletions(-)

Index: 2.6.25-rc5-mm1/include/linux/devpts\_fs.h

```
=====
--- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:04:26.000000000 -0700
+++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:04:31.000000000 -0700
@@ -14,6 +14,17 @@
#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

```

```

Index: 2.6.25-rc5-mm1/fs/devpts/inode.c

```

```

=====

```

```

--- 2.6.25-rc5-mm1.orig/fs/devpts/inode.c 2008-03-24 20:04:26.000000000 -0700

```

```

+++ 2.6.25-rc5-mm1/fs/devpts/inode.c 2008-03-24 20:04:31.000000000 -0700

```

```

@@ -28,12 +28,8 @@

```

```

#define DEVPTS_DEFAULT_MODE 0600

```

```

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;
-

```

```

static struct {
    int setuid;
    int setgid;
@@ -54,6 +50,15 @@ static match_table_t tokens = {
    {Opt_err, NULL}
};

```

```

+struct pts_namespace init_pts_ns = {
+ .kref = {
+ .refcount = ATOMIC_INIT(2),
+ },
+ .allocated_ptys = IDR_INIT(init_pts_ns.allocated_ptys),
+ .mnt = NULL,
+};
+
+
static int devpts_remount(struct super_block *sb, int *flags, char *data)
{
    char *p;
@@ -140,7 +145,7 @@ 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;

@@ -150,17 +155,82 @@ 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)
+{
+  /*
+   * new_pts_ns() mounts the pts namespace and free_pts_ns()
+   * drops the reference to the mount. i.e the s_fs_inf is
+   * cleared and vfsmnt is releasand _before_ pts_namespace
+   * is freed.
+   *
+   * So we don't need a reference to the pts_namespace here
+   * (Getting a reference here will also cause circular reference).
+   */
+  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;
+  + struct pts_namespace *ns;
+  + int err;
+  +
+  + /* hereafter we're very similar to proc_get_sb */

```

```

+ if (flags & MS_KERNMOUNT)
+ ns = data;
+ else
+ ns = &init_pts_ns;
+
+ /* hereafter we're very similar to get_sb_nodev */
+ sb = sget(fs_type, devpts_test_sb, devpts_set_sb, ns);
+ 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;
+ ns->mnt = mnt;
+
+ return simple_set_mnt(mnt, sb);
+}
+
+static void devpts_kill_sb(struct super_block *sb)
+{
+ sb->s_fs_info = NULL;
+ kill_anon_super(sb);
+ }

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,
};

/*
@@ -168,10 +238,9 @@ 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));
}
@@ -180,14 +249,15 @@ int devpts_new_index(void)
{
    int index;
    int idr_ret;
+ struct pts_namespace *pts_ns = &init_pts_ns;

    retry:
- if (!idr_pre_get(&allocated_ptys, GFP_KERNEL)) {
+ if (!idr_pre_get(&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(&pts_ns->allocated_ptys, NULL, &index);
    if (idr_ret < 0) {
        up(&allocated_ptys_lock);
        if (idr_ret == -EAGAIN)
@@ -196,7 +266,7 @@ retry:
    }

    if (index >= pty_limit) {
- idr_remove(&allocated_ptys, index);
+ idr_remove(&pts_ns->allocated_ptys, index);
        up(&allocated_ptys_lock);
        return -EIO;
    }
@@ -206,8 +276,10 @@ retry:

void devpts_kill_index(int idx)
{
+ struct pts_namespace *pts_ns = &init_pts_ns;
+
    down(&allocated_ptys_lock);
- idr_remove(&allocated_ptys, idx);
+ idr_remove(&pts_ns->allocated_ptys, idx);
    up(&allocated_ptys_lock);
}

@@ -217,12 +289,26 @@ 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;
+ struct pts_namespace *pts_ns = &init_pts_ns;

    /* 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 = pts_ns->mnt;
+ root = mnt->mnt_root;
+
+ mutex_lock(&root->d_inode->i_mutex);
+ inode = idr_find(&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;

@@ -232,23 +318,29 @@ 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(&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 = init_pts_ns.mnt;
+
+ dentry = get_node(mnt->mnt_root, number);
+
  tty = NULL;
  if (!IS_ERR(dentry)) {
    if (dentry->d_inode)
@@ -256,14 +348,19 @@ 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;
+
+ root = init_pts_ns.mnt->mnt_root;
+
+ dentry = get_node(root, number);

  if (!IS_ERR(dentry)) {
    struct inode *inode = dentry->d_inode;
@@ -274,24 +371,31 @@ 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, &init_pts_ns);
+ if (IS_ERR(mnt))
+ err = PTR_ERR(mnt);
+ else
+ 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)

---

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

---



---

Subject: [PATCH 4/7] Implement get\_pts\_ns() and put\_pts\_ns()  
Posted by [Sukadev Bhattiprolu](#) on Tue, 25 Mar 2008 04:25:07 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
Subject: [PATCH 4/7]: Implement get\_pts\_ns() and put\_pts\_ns()

Implement get\_pts\_ns() and put\_pts\_ns() interfaces.

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

---

include/linux/devpts\_fs.h | 21 ++++++++  
1 file changed, 20 insertions(+), 1 deletion(-)

Index: 2.6.25-rc5-mm1/include/linux/devpts\_fs.h

```

=====
--- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:04:31.000000000 -0700

```

```

+++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:05:05.000000000 -0700
@@ -27,13 +27,26 @@ struct pts_namespace {
extern struct pts_namespace init_pts_ns;

#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 */

+static inline void free_pts_ns(struct kref *ns_kref) { }
+
+static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
+{
+ if (ns && (ns != &init_pts_ns))
+ kref_get(&ns->kref);
+ return ns;
+}
+static inline void put_pts_ns(struct pts_namespace *ns)
+{
+ if (ns && (ns != &init_pts_ns))
+ kref_put(&ns->kref, free_pts_ns);
+}
+
+
+/* Dummy stubs in the no-pty case */
@@ -43,6 +56,12 @@ static inline int devpts_pty_new(struct
static inline struct tty_struct *devpts_get_tty(int number) { return NULL; }
static inline void devpts_pty_kill(int number) { }

+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) { }
#endif

```

---

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

---

Subject: [PATCH 5/7]: Determine pts\_ns from a pty's inode.  
Posted by [Sukadev Bhattiprolu](#) on Tue, 25 Mar 2008 04:25:41 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
Subject: [PATCH 5/7]: Determine pts\_ns from a pty's inode.

The devpts interfaces currently operate on a specific pts namespace which they get from the 'current' task.

With implementation of containers and cloning of PTS namespaces, we want to be able to access PTYs in a child-pts-ns from a parent-pts-ns. For instance we could bind-mount and pivot-root the child container on '/vserver/vserver1' and then access the "pts/0" of 'vserver1' using

```
$ echo foo > /vserver/vserver1/dev/pts/0
```

The task doing the above 'echo' could be in parent-pts-ns. So we find the 'pts-ns' of the above file from the inode representing the above file rather than from the 'current' task.

Note that we need to find and hold a reference to the pts\_ns to prevent the pts\_ns from being freed while it is being accessed from 'outside'.

This patch implements, 'pts\_ns\_from\_inode()' which returns the pts\_ns using 'inode->i\_sb->s\_fs\_info'.

Since, the 'inode' information is not visible inside devpts code itself, this patch modifies the tty driver code to determine the pts\_ns and passes it into devpts.

TODO:

What is the expected behavior when '/dev/tty' or '/dev/ptmx' are accessed from parent-pts-ns. i.e:

```
$ echo "foobar" > /vserver/vserver1/dev/tty)
```

This patch currently ignores the '/vserver/vserver1' part (that seemed to be the easiest to do :-). So opening /dev/ptmx from even the child pts-ns will create a pty in the \_PARENT\_ pts-ns.

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

---

```
drivers/char/pty.c      | 2 -  
drivers/char/tty_io.c  | 86 ++++++++++++++++++++++++++++++++++++++-----  
fs/devpts/inode.c      | 19 +++-----  
include/linux/devpts_fs.h | 42 +++++++++++++++++++++++++++++++++-----  
4 files changed, 119 insertions(+), 30 deletions(-)
```



Index: 2.6.25-rc5-mm1/include/linux/devpts\_fs.h

```
=====
--- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:05:05.000000000 -0700
+++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:08:33.000000000 -0700
@@ -17,6 +17,7 @@
#include <linux/nsproxy.h>
#include <linux/kref.h>
#include <linux/idr.h>
+#include <linux/fs.h>

struct pts_namespace {
    struct kref kref;
@@ -26,12 +27,43 @@ struct pts_namespace {

extern struct pts_namespace init_pts_ns;

+#define DEVPTS_SUPER_MAGIC 0x1cd1
+static inline struct pts_namespace *pts_ns_from_inode(struct inode *inode)
+{
+ /*
+  * Need this bug-on for now to catch any cases in tty_open()
+  * or release_dev() I may have missed.
+  */
+ BUG_ON(inode->i_sb->s_magic != DEVPTS_SUPER_MAGIC);
+
+ /*
+  * If we have a valid inode, we already have a reference to
+  * mount-point. Since there is a single super-block for the
+  * devpts mount, i_sb->s_fs_info cannot go to NULL. So we
+  * should not need a lock here.
+  */
+
+ return (struct pts_namespace *)inode->i_sb->s_fs_info;
+}
+
+static inline struct pts_namespace *current_pts_ns(void)
+{
+ return &init_pts_ns;
+}
+
+#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 */
+int devpts_new_index(struct pts_namespace *pts_ns);
```

```

+void devpts_kill_index(struct pts_namespace *pts_ns, int idx);
+
+/* mknod in devpts */
+int devpts_pty_new(struct pts_namespace *pts_ns, struct tty_struct *tty);
+
+/* get tty structure */
+struct tty_struct *devpts_get_tty(struct pts_namespace *pts_ns, int number);
+
+/* unlink */
+void devpts_pty_kill(struct pts_namespace *pts_ns, int number);

static inline void free_pts_ns(struct kref *ns_kref) { }

```

Index: 2.6.25-rc5-mm1/drivers/char/tty\_io.c

```

=====
--- 2.6.25-rc5-mm1.orig/drivers/char/tty_io.c 2008-03-24 20:04:26.000000000 -0700
+++ 2.6.25-rc5-mm1/drivers/char/tty_io.c 2008-03-24 20:08:15.000000000 -0700
@@ -2064,8 +2064,8 @@ static void tty_line_name(struct tty_dri
 * relaxed for the (most common) case of reopening a tty.
 */

-static int init_dev(struct tty_driver *driver, int idx,
- struct tty_struct **ret_tty)
+static int init_dev(struct tty_driver *driver, struct pts_namespace *pts_ns,
+ int idx, struct tty_struct **ret_tty)
{
    struct tty_struct *tty, *o_tty;
    struct ktermios *tp, **tp_loc, *o_tp, **o_tp_loc;
@@ -2074,7 +2074,7 @@ static int init_dev(struct tty_driver *d

    /* check whether we're reopening an existing tty */
    if (driver->flags & TTY_DRIVER_DEVPTS_MEM) {
- tty = devpts_get_tty(idx);
+ tty = devpts_get_tty(pts_ns, idx);
    /*
     * If we don't have a tty here on a slave open, it's because
     * the master already started the close process and there's
@@ -2361,6 +2361,43 @@ static void release_tty(struct tty_struct
    }

    /*
+ * When opening /dev/tty and /dev/ptmx, use the pts-ns of the calling
+ * process. For any other pts device, use the pts-ns, in which the
+ * device was created. This latter case is needed when the pty is
+ * being accessed from a parent container.
+ *
+ * Eg: Suppose the user used bind-mount and pivot-root to mount a
+ * child- container's root on /vs/vs1. Then "/vs/vs1/dev/pts/0"

```

```

+ * in parent container and "/dev/pts/0" in child container would
+ * refer to the same device.
+ *
+ * When parent-container opens, "/vs/vs1/dev/pts/0" we find and
+ * grab/drop reference to child container's pts-ns (using @filp).
+ */
+struct pts_namespace *pty_pts_ns(struct tty_driver *driver, struct inode *inode)
+{
+
+ int devpts;
+ int pty_master;
+ dev_t dev;
+ struct pts_namespace *pts_ns;
+
+ devpts = (driver->flags & TTY_DRIVER_DEVPTS_MEM) != 0;
+ pty_master = (driver->type == TTY_DRIVER_TYPE_PTY &&
+   driver->subtype == PTY_TYPE_MASTER);
+
+ pts_ns = NULL;
+ if (devpts) {
+   dev = inode->i_rdev;
+   if (pty_master || dev == MKDEV(TTYAUX_MAJOR, 0))
+     pts_ns = current_pts_ns();
+   else
+     pts_ns = pts_ns_from_inode(inode);
+ }
+ return pts_ns;
+}
+
+/*
+ * Even releasing the tty structures is a tricky business.. We have
+ * to be very careful that the structures are all released at the
+ * same time, as interrupts might otherwise get the wrong pointers.
@@ -2376,10 +2413,12 @@ static void release_dev(struct file *fil
   int idx;
   char buf[64];
   unsigned long flags;
+ struct pts_namespace *pts_ns;
+ struct inode *inode;

+ inode = filp->f_path.dentry->d_inode;
+ tty = (struct tty_struct *)filp->private_data;
- if (tty_paranoia_check(tty, filp->f_path.dentry->d_inode,
-   "release_dev"))
+ if (tty_paranoia_check(tty, inode, "release_dev"))
   return;

   check_tty_count(tty, "release_dev");

```

```

@@ -2391,6 +2430,7 @@ static void release_dev(struct file *fil
    tty->driver->subtype == PTY_TYPE_MASTER);
    devpts = (tty->driver->flags & TTY_DRIVER_DEVPTS_MEM) != 0;
    o_tty = tty->link;
+ pts_ns = pty_pts_ns(tty->driver, inode);

#ifdef TTY_PARANOIA_CHECK
    if (idx < 0 || idx >= tty->driver->num) {
@@ -2633,8 +2673,13 @@ static void release_dev(struct file *fil
    release_tty(tty, idx);

    /* Make this pty number available for reallocation */
- if (devpts)
- devpts_kill_index(idx);
+ if (devpts) {
+ devpts_kill_index(pts_ns, idx);
+ /*
+  * Drop reference got in init_dev()
+  */
+ put_pts_ns(pts_ns);
+ }
}

/**
@@ -2666,6 +2711,7 @@ static int tty_open(struct inode *inode,
    int index;
    dev_t device = inode->i_rdev;
    unsigned short saved_flags = filp->f_flags;
+ struct pts_namespace *pts_ns;

    nonseekable_open(inode, filp);

@@ -2715,7 +2761,20 @@ retry_open:
    return -ENODEV;
}
got_driver:
- retval = init_dev(driver, index, &tty);
+
+ /*
+  * What pts-ns do we want to use when opening "/dev/tty" ?
+  * Sounds like current_pts_ns(), but what should happen
+  * if parent pts ns does:
+  *
+  * echo foo > /vs/vs1/dev/tty
+  *
+  * (See Serge's setupvs1 script for the /vs/vs1...)
+  */
+ pts_ns = pty_pts_ns(driver, inode);

```

```

+ get_pts_ns(pts_ns);
+
+ retval = init_dev(driver, pts_ns, index, &tty);
  mutex_unlock(&tty_mutex);
  if (retval)
    return retval;
@@ -2790,16 +2849,19 @@ static int ptmx_open(struct inode *inode
  struct tty_struct *tty;
  int retval;
  int index;
+ struct pts_namespace *pts_ns;

  nonseekable_open(inode, filp);

+ pts_ns = current_pts_ns();
+
  /* find a device that is not in use. */
- index = devpts_new_index();
+ index = devpts_new_index(pts_ns);
  if (index < 0)
    return index;

  mutex_lock(&tty_mutex);
- retval = init_dev(ptm_driver, index, &tty);
+ retval = init_dev(ptm_driver, pts_ns, index, &tty);
  mutex_unlock(&tty_mutex);

  if (retval)
@@ -2809,7 +2871,7 @@ static int ptmx_open(struct inode *inode
  filp->private_data = tty;
  file_move(filp, &tty->tty_files);

- retval = devpts_pty_new(tty->link);
+ retval = devpts_pty_new(pts_ns, tty->link);
  if (retval)
    goto out1;

@@ -2821,7 +2883,7 @@ out1:
  release_dev(filp);
  return retval;
out:
- devpts_kill_index(index);
+ devpts_kill_index(pts_ns, index);
  return retval;
}
#endif

```

Index: 2.6.25-rc5-mm1/fs/devpts/inode.c

=====

```

--- 2.6.25-rc5-mm1.orig/fs/devpts/inode.c 2008-03-24 20:04:31.000000000 -0700
+++ 2.6.25-rc5-mm1/fs/devpts/inode.c 2008-03-24 20:08:33.000000000 -0700
@@ -23,8 +23,6 @@
#include <linux/fsnotify.h>
#include <linux/seq_file.h>

-#define DEVPTS_SUPER_MAGIC 0x1cd1
-
#define DEVPTS_DEFAULT_MODE 0600

extern int pty_limit; /* Config limit on Unix98 ptys */
@@ -245,11 +243,10 @@ static struct dentry *get_node(struct de
return lookup_one_len(s, root, sprintf(s, "%d", num));
}

-int devpts_new_index(void)
+int devpts_new_index(struct pts_namespace *pts_ns)
{
    int index;
    int idr_ret;
- struct pts_namespace *pts_ns = &init_pts_ns;

    retry:
    if (!idr_pre_get(&pts_ns->allocated_ptys, GFP_KERNEL)) {
@@ -274,16 +271,15 @@ retry:
return index;
}

-void devpts_kill_index(int idx)
+void devpts_kill_index(struct pts_namespace *pts_ns, int idx)
{
- struct pts_namespace *pts_ns = &init_pts_ns;

    down(&allocated_ptys_lock);
    idr_remove(&pts_ns->allocated_ptys, idx);
    up(&allocated_ptys_lock);
}

-int devpts_pty_new(struct tty_struct *tty)
+int devpts_pty_new(struct pts_namespace *pts_ns, struct tty_struct *tty)
{
    int number = tty->index; /* tty layer puts index from devpts_new_index() in here */
    struct tty_driver *driver = tty->driver;
@@ -292,7 +288,6 @@ int devpts_pty_new(struct tty_struct *tt
struct dentry *root;
struct vfsmount *mnt;
struct inode *inode;
- struct pts_namespace *pts_ns = &init_pts_ns;

```

```

/* We're supposed to be given the slave end of a pty */
BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
@@ -331,13 +326,13 @@ int devpts_pty_new(struct tty_struct *tt
    return 0;
}

```

```

-struct tty_struct *devpts_get_tty(int number)
+struct tty_struct *devpts_get_tty(struct pts_namespace *pts_ns, int number)
{
    struct vfsmount *mnt;
    struct dentry *dentry;
    struct tty_struct *tty;

```

```

- mnt = init_pts_ns.mnt;
+ mnt = pts_ns->mnt;

```

```

    dentry = get_node(mnt->mnt_root, number);

```

```

@@ -353,12 +348,12 @@ struct tty_struct *devpts_get_tty(int nu
    return tty;
}

```

```

-void devpts_pty_kill(int number)
+void devpts_pty_kill(struct pts_namespace *pts_ns, int number)
{
    struct dentry *dentry;
    struct dentry *root;

```

```

- root = init_pts_ns.mnt->mnt_root;
+ root = pts_ns->mnt->mnt_root;

```

```

    dentry = get_node(root, number);

```

Index: 2.6.25-rc5-mm1/drivers/char/pty.c

```

=====
--- 2.6.25-rc5-mm1.orig/drivers/char/pty.c 2008-03-24 20:02:57.000000000 -0700
+++ 2.6.25-rc5-mm1/drivers/char/pty.c 2008-03-24 20:08:15.000000000 -0700
@@ -59,7 +59,7 @@ static void pty_close(struct tty_struct
    set_bit(TTY_OTHER_CLOSED, &tty->flags);
#ifdef CONFIG_UNIX98_PTYS
    if (tty->driver == ptm_driver)
-   devpts_pty_kill(tty->index);
+   devpts_pty_kill(current_pts_ns(), tty->index);
#endif
    tty_vhangup(tty->link);
}

```

---

Subject: [PATCH 6/7]: Check for user-space mount of /dev/pts  
Posted by [Sukadev Bhattiprolu](#) on Tue, 25 Mar 2008 04:26:14 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
Subject: [PATCH 6/7]: Check for user-space mount of /dev/pts

When the pts namespace is cloned, the /dev/pts is not useful unless it is remounted from the user space.

If user-space clones pts namespace but does not remount /dev/pts, it would end up using the /dev/pts mount from parent-pts-ns but allocate the pts indices from current pts ns.

This patch (hack ?) prevents creation of PTYs in user space unless user-space mounts /dev/pts.

(While this patch can be folded into others, keeping this separate for now for easier review (and to highlight the hack :-)

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

---

fs/devpts/inode.c | 25 ++++++  
include/linux/devpts\_fs.h | 20 ++++++  
2 files changed, 42 insertions(+), 3 deletions(-)

Index: 2.6.25-rc5-mm1/include/linux/devpts\_fs.h

```
=====
--- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:08:33.000000000 -0700
+++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:08:57.000000000 -0700
@@ -23,6 +23,7 @@ struct pts_namespace {
     struct kref kref;
     struct idr allocated_ptys;
     struct vfsmount *mnt;
+ int user_mounted;
 };

extern struct pts_namespace init_pts_ns;
@@ -30,6 +31,8 @@ extern struct pts_namespace init_pts_ns;
#define DEVPTS_SUPER_MAGIC 0x1cd1
static inline struct pts_namespace *pts_ns_from_inode(struct inode *inode)
{
+ struct pts_namespace *ns;
```



```

+
+ /*
+  * Need this bug-on for now to catch any cases in tty_open()
+  * or release_dev() I may have missed.
@@ -43,7 +46,22 @@ static inline struct pts_namespace *pts_
+  * should not need a lock here.
+ */

- return (struct pts_namespace *)inode->i_sb->s_fs_info;
+ ns = (struct pts_namespace *)inode->i_sb->s_fs_info;
+
+ /*
+  * If user-space did not mount pts ns after cloning pts namespace,
+  * the child process would end up accessing devpts mount of the
+  * parent but use allocated_ptys from the cloned pts ns.
+  *
+  * This check prevents creating ptys unless user-space mounts
+  * devpts in the new pts namespace.
+  *
+  * Is there a cleaner way to prevent this ?
+  */
+ if (!ns->user_mounted)
+ return NULL;
+
+ return ns;
+ }

```

static inline struct pts\_namespace \*current\_pts\_ns(void)

Index: 2.6.25-rc5-mm1/fs/devpts/inode.c

```

=====
--- 2.6.25-rc5-mm1.orig/fs/devpts/inode.c 2008-03-24 20:08:33.000000000 -0700
+++ 2.6.25-rc5-mm1/fs/devpts/inode.c 2008-03-24 20:08:57.000000000 -0700
@@ -201,8 +201,11 @@ static int devpts_get_sb(struct file_sys
+ if (IS_ERR(sb))
+ return PTR_ERR(sb);

- if (sb->s_root)
+ if (sb->s_root) {
+ if (!(flags & MS_KERNMOUNT))
+ ns->user_mounted = 1;
+ return simple_set_mnt(mnt, sb);
+ }

sb->s_flags = flags;
err = devpts_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
@@ -248,6 +251,10 @@ int devpts_new_index(struct pts_namespac
int index;
int idr_ret;

```

```

+ if (!pts_ns || !pts_ns->user_mounted) {
+   printk(KERN_ERR "devpts_new_index() without user_mount\n");
+   return -ENOSYS;
+ }
retry:
  if (!idr_pre_get(&pts_ns->allocated_ptys, GFP_KERNEL)) {
    return -ENOMEM;
@@ -273,7 +280,7 @@ retry:

void devpts_kill_index(struct pts_namespace *pts_ns, int idx)
{
-
+ BUG_ON(!pts_ns->user_mounted);
  down(&allocated_ptys_lock);
  idr_remove(&pts_ns->allocated_ptys, idx);
  up(&allocated_ptys_lock);
@@ -293,6 +300,11 @@ int devpts_pty_new( struct pts_namespace
  BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
  BUG_ON(driver->subtype != PTY_TYPE_SLAVE);

+ if (!pts_ns || !pts_ns->user_mounted) {
+   printk(KERN_ERR "devpts_pty_new() without user_mount\n");
+   return -ENOSYS;
+ }
+
  mnt = pts_ns->mnt;
  root = mnt->mnt_root;

@@ -332,6 +344,11 @@ struct tty_struct *devpts_get_tty(struct
  struct dentry *dentry;
  struct tty_struct *tty;

+ if (!pts_ns || !pts_ns->user_mounted) {
+   printk(KERN_ERR "devpts_get_tty() without user_mount\n");
+   return ERR_PTR(-ENOSYS);
+ }
+
  mnt = pts_ns->mnt;

  dentry = get_node(mnt->mnt_root, number);
@@ -353,6 +370,10 @@ void devpts_pty_kill(struct pts_namespac
  struct dentry *dentry;
  struct dentry *root;

+ if (!pts_ns || !pts_ns->user_mounted) {
+   printk(KERN_ERR "devpts_pty_kill() without user_mount\n");
+   BUG_ON(1);

```

```
+ }
root = pts_ns->mnt->mnt_root;

dentry = get_node(root, number);
```

---

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

---

---

Subject: [PATCH 7/7]: Enable cloning PTY namespaces  
Posted by [Sukadev Bhattiprolu](#) on Tue, 25 Mar 2008 04:27:19 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

From: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
Subject: [PATCH 7/7]: Enable cloning PTY namespaces

Enable cloning PTY namespaces.

Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
Signed-off-by: Serge Hallyn <serue@us.ibm.com>  
Signed-off-by: Matt Helsley <matthlrc@us.ibm.com>

```
---
fs/devpts/inode.c      | 40 ++++++
include/linux/devpts_fs.h | 22 ++++++
include/linux/init_task.h | 1 +
include/linux/nsproxy.h | 2 ++
include/linux/sched.h   | 1 +
kernel/fork.c           | 2 +-
kernel/nsproxy.c        | 17 ++++++
7 files changed, 79 insertions(+), 6 deletions(-)
```

Index: 2.6.25-rc5-mm1/include/linux/sched.h

```
=====
--- 2.6.25-rc5-mm1.orig/include/linux/sched.h 2008-03-24 20:02:57.000000000 -0700
+++ 2.6.25-rc5-mm1/include/linux/sched.h 2008-03-24 20:12:56.000000000 -0700
@@ -28,6 +28,7 @@
#define CLONE_NEWPID 0x20000000 /* New pid namespace */
#define CLONE_NEWNET 0x40000000 /* New network namespace */
#define CLONE_IO 0x80000000 /* Clone io context */
+#define CLONE_NEWPTS 0x0000000200000000ULL /* Clone pts ns */
```

```
/*
 * Scheduling policies
Index: 2.6.25-rc5-mm1/include/linux/nsproxy.h
```

```
=====
--- 2.6.25-rc5-mm1.orig/include/linux/nsproxy.h 2008-03-24 20:02:57.000000000 -0700
```

```
+++ 2.6.25-rc5-mm1/include/linux/nsproxy.h 2008-03-24 20:12:56.000000000 -0700
@@ -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: 2.6.25-rc5-mm1/include/linux/init\_task.h

```
=====
--- 2.6.25-rc5-mm1.orig/include/linux/init_task.h 2008-03-24 20:02:57.000000000 -0700
+++ 2.6.25-rc5-mm1/include/linux/init_task.h 2008-03-24 20:12:56.000000000 -0700
@@ -78,6 +78,7 @@ 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: 2.6.25-rc5-mm1/include/linux/devpts\_fs.h

```
=====
--- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:08:57.000000000 -0700
+++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:12:56.000000000 -0700
@@ -66,7 +66,7 @@ static inline struct pts_namespace *pts_

static inline struct pts_namespace *current_pts_ns(void)
{
- return &init_pts_ns;
+ return current->nsproxy->pts_ns;
}
```

```
@@ -83,7 +83,8 @@ struct tty_struct *devpts_get_tty(struct
/* unlink */
void devpts_pty_kill(struct pts_namespace *pts_ns, int number);

-static inline void free_pts_ns(struct kref *ns_kref) { }
+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)
{
@@ -97,6 +98,15 @@ static inline void put_pts_ns(struct pts
    kref_put(&ns->kref, free_pts_ns);
}

+static inline struct pts_namespace *copy_pts_ns(u64 flags,
+        struct pts_namespace *old_ns)
+{
+    if (flags & CLONE_NEWPTS)
+        return new_pts_ns();
+    else
+        return get_pts_ns(old_ns);
+}
+
#else

/* Dummy stubs in the no-pty case */
@@ -112,6 +122,14 @@ static inline struct pts_namespace *get_
}

static inline void put_pts_ns(struct pts_namespace *ns) { }
+
+static inline struct pts_namespace *copy_pts_ns(u64 flags,
+        struct pts_namespace *old_ns)
+{
+    if (flags & CLONE_NEWPTS)
+        return ERR_PTR(-EINVAL);
+    return old_ns;
+}
#endif

```

Index: 2.6.25-rc5-mm1/fs/devpts/inode.c

```

=====
--- 2.6.25-rc5-mm1.orig/fs/devpts/inode.c 2008-03-24 20:08:57.000000000 -0700
+++ 2.6.25-rc5-mm1/fs/devpts/inode.c 2008-03-24 20:14:20.000000000 -0700
@@ -27,6 +27,7 @@

```

```

extern int pty_limit; /* Config limit on Unix98 ptys */
static DECLARE_MUTEX(allocated_ptys_lock);
+static struct file_system_type devpts_fs_type;

static struct {
    int setuid;
@@ -56,6 +57,43 @@ struct pts_namespace init_pts_ns = {

```

```

.mnt = NULL,
};

+struct pts_namespace *new_pts_ns(void)
+{
+ struct pts_namespace *ns;
+
+ ns = kmalloc(sizeof(*ns), GFP_KERNEL);
+ if (!ns)
+ return ERR_PTR(-ENOMEM);
+
+ kref_init(&ns->kref);
+
+ 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);
+ ns->user_mounted = 0;
+
+ return ns;
+}
+
+void free_pts_ns(struct kref *ns_kref)
+{
+ struct pts_namespace *ns;
+
+ ns = container_of(ns_kref, struct pts_namespace, kref);
+ 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)
{
@@ -194,7 +232,7 @@ static int devpts_get_sb(struct file_sys
if (flags & MS_KERNMOUNT)
ns = data;
else
- ns = &init_pts_ns;

```

```

+ ns = current_pts_ns();

/* hereafter we're very similar to get_sb_nodev */
sb = sget(fs_type, devpts_test_sb, devpts_set_sb, ns);
Index: 2.6.25-rc5-mm1/kernel/fork.c
=====
--- 2.6.25-rc5-mm1.orig/kernel/fork.c 2008-03-24 20:02:57.000000000 -0700
+++ 2.6.25-rc5-mm1/kernel/fork.c 2008-03-24 20:12:56.000000000 -0700
@@ -1713,7 +1713,7 @@ static long do_unshare(u64 unshare_flags
    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: 2.6.25-rc5-mm1/kernel/nsproxy.c
=====
--- 2.6.25-rc5-mm1.orig/kernel/nsproxy.c 2008-03-24 20:02:57.000000000 -0700
+++ 2.6.25-rc5-mm1/kernel/nsproxy.c 2008-03-24 20:12:56.000000000 -0700
@@ -21,6 +21,7 @@
#include <linux/utsname.h>
#include <linux/pid_namespace.h>
#include <net/net_namespace.h>
+#include <linux/devpts_fs.h>
#include <linux/ipc_namespace.h>

static struct kmem_cache *nsproxy_cache;
@@ -93,8 +94,17 @@ 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);
@@ -131,7 +141,8 @@ int copy_namespaces(u64 flags, struct ta

```

```

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)) {
@@ -170,6 +181,8 @@ 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);
}
@@ -184,7 +197,7 @@ int unshare_nsproxy_namespaces(u64 unsha
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>

---



---

Subject: Re: [PATCH 6/7]: Check for user-space mount of /dev/pts

Posted by [Pavel Emelianov](#) on Tue, 25 Mar 2008 07:46:55 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:

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

> Subject: [PATCH 6/7]: Check for user-space mount of /dev/pts

>

> When the pts namespace is cloned, the /dev/pts is not useful unless it  
> is remounted from the user space.

>

> If user-space clones pts namespace but does not remount /dev/pts, it  
> would end up using the /dev/pts mount from parent-pts-ns but allocate  
> the pts indices from current pts ns.

>



```
> This patch (hack ?) prevents creation of PTYs in user space unless
> user-space mounts /dev/pts.
>
> (While this patch can be folded into others, keeping this separate
> for now for easier review (and to highlight the hack :-))
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> fs/devpts/inode.c      | 25 ++++++
> include/linux/devpts_fs.h | 20 ++++++
> 2 files changed, 42 insertions(+), 3 deletions(-)
>
> Index: 2.6.25-rc5-mm1/include/linux/devpts_fs.h
> =====
```

[snip]

```
> =====
> --- 2.6.25-rc5-mm1.orig/fs/devpts/inode.c 2008-03-24 20:08:33.000000000 -0700
> +++ 2.6.25-rc5-mm1/fs/devpts/inode.c 2008-03-24 20:08:57.000000000 -0700
> @@ -201,8 +201,11 @@ static int devpts_get_sb(struct file_sys
>  if (IS_ERR(sb))
>   return PTR_ERR(sb);
>
> - if (sb->s_root)
> + if (sb->s_root) {
> + if (!(flags & MS_KERNMOUNT))
> + ns->user_mounted = 1;
```

What if user space umounts this back? Won't this break?

Thanks,  
Pavel

---

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

---

---

Subject: Re: [PATCH 0/7][v2] Cloning PTS namespace  
Posted by [Pavel Emelianov](#) on Tue, 25 Mar 2008 07:51:15 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

sukadev@us.ibm.com wrote:  
> 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 1/7]: Propagate error code from devpts\_pty\_new

> [PATCH 2/7]: Factor out PTY index allocation

> [PATCH 3/7]: Enable multiple mounts of /dev/pts

> [PATCH 4/7]: Implement get\_pts\_ns() and put\_pts\_ns()

> [PATCH 5/7]: Determine pts\_ns from a pty's inode.

> [PATCH 6/7]: Check for user-space mount of /dev/pts

> [PATCH 7/7]: Enable cloning PTY namespaces

>

> Todo:

> - This patchset depends on availability of additional clone flags.

> and relies on on Cedric's clone64 patchset.

But this set was not even reviewed, wasn't it? Does anybody  
know what are the plans about exhausted flags?

Thanks,  
Pavel

---

Containers mailing list

Containers@lists.linux-foundation.org

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

---

Subject: Re: [PATCH 6/7]: Check for user-space mount of /dev/pts

Posted by [Alexey Dobriyan](#) on Tue, 25 Mar 2008 09:40:19 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

On Tuesday 25 March 2008 07:26:14 sukadev@us.ibm.com wrote:

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

> Subject: [PATCH 6/7]: Check for user-space mount of /dev/pts

>

> When the pts namespace is cloned, the /dev/pts is not useful unless it  
> is remounted from the user space.

>

> If user-space clones pts namespace but does not remount /dev/pts, it  
> would end up using the /dev/pts mount from parent-pts-ns but allocate  
> the pts indices from current pts ns.

>

> This patch (hack ?) prevents creation of PTYs in user space unless  
> user-space mounts /dev/pts.

This is absolutely horrible and shouldn't go anywhere.

---

---

Subject: Re: [PATCH 0/7][v2] Cloning PTS namespace

Posted by [serue](#) on Tue, 25 Mar 2008 14:42:52 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Quoting Pavel Emelyanov (xemul@openvz.org):

> sukadev@us.ibm.com wrote:

> > 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 1/7]: Propagate error code from devpts\_pty\_new

> > [PATCH 2/7]: Factor out PTY index allocation

> > [PATCH 3/7]: Enable multiple mounts of /dev/pts

> > [PATCH 4/7]: Implement get\_pts\_ns() and put\_pts\_ns()

> > [PATCH 5/7]: Determine pts\_ns from a pty's inode.

> > [PATCH 6/7]: Check for user-space mount of /dev/pts

> > [PATCH 7/7]: Enable cloning PTY namespaces

> >

> > Todo:

> > - This patchset depends on availability of additional clone flags.

> > and relies on on Cedric's clone64 patchset.

>

> But this set was not even reviewed, wasn't it?

Well it was sent out! :) Last time was Feb 11. There'd been no responses so while Cedric has been maintaining it, I suspect he hasn't wanted to gratuitously spam the list with countless resends.

> Does anybody

> know what are the plans about exhausted flags?

hopefully "go with clone64/unshare64" :)

-serge

---

Containers mailing list

Containers@lists.linux-foundation.org

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

---

---

Subject: Re: [PATCH 6/7]: Check for user-space mount of /dev/pts

Posted by [serue](#) on Tue, 25 Mar 2008 14:54:48 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):

```
>
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [PATCH 6/7]: Check for user-space mount of /dev/pts
>
> When the pts namespace is cloned, the /dev/pts is not useful unless it
> is remounted from the user space.
>
> If user-space clones pts namespace but does not remount /dev/pts, it
> would end up using the /dev/pts mount from parent-pts-ns but allocate
> the pts indices from current pts ns.
```

So why not use the allocated\_ptys from the parent ptsns? It's what userspace asked for and it's safe to do.

```
> This patch (hack ?) prevents creation of PTYs in user space unless
> user-space mounts /dev/pts.
>
> (While this patch can be folded into others, keeping this separate
> for now for easier review (and to highlight the hack :-))
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> fs/devpts/inode.c      | 25 ++++++++
> include/linux/devpts_fs.h | 20 +++++
> 2 files changed, 42 insertions(+), 3 deletions(-)
>
> Index: 2.6.25-rc5-mm1/include/linux/devpts_fs.h
> =====
> --- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:08:33.000000000 -0700
> +++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:08:57.000000000 -0700
> @@ -23,6 +23,7 @@ struct pts_namespace {
> struct kref kref;
> struct idr allocated_ptys;
> struct vfsmount *mnt;
> + int user_mounted;
> };
>
> extern struct pts_namespace init_pts_ns;
> @@ -30,6 +31,8 @@ extern struct pts_namespace init_pts_ns;
> #define DEVPTS_SUPER_MAGIC 0x1cd1
> static inline struct pts_namespace *pts_ns_from_inode(struct inode *inode)
> {
> + struct pts_namespace *ns;
> +
> /*
> * Need this bug-on for now to catch any cases in tty_open()
> * or release_dev() I may have missed.
```

```

> @@ -43,7 +46,22 @@ static inline struct pts_namespace *pts_
> * should not need a lock here.
> */
>
> - return (struct pts_namespace *)inode->i_sb->s_fs_info;
> + ns = (struct pts_namespace *)inode->i_sb->s_fs_info;
> +
> + /*
> + * If user-space did not mount pts ns after cloning pts namespace,
> + * the child process would end up accessing devpts mount of the
> + * parent but use allocated_ptys from the cloned pts ns.
> + *
> + * This check prevents creating ptys unless user-space mounts
> + * devpts in the new pts namespace.
> + *
> + * Is there a cleaner way to prevent this ?
> + */
> + if (!ns->user_mounted)
> + return NULL;
> +
> + return ns;
> }
>
> static inline struct pts_namespace *current_pts_ns(void)
> Index: 2.6.25-rc5-mm1/fs/devpts/inode.c
> =====
> --- 2.6.25-rc5-mm1.orig/fs/devpts/inode.c 2008-03-24 20:08:33.000000000 -0700
> +++ 2.6.25-rc5-mm1/fs/devpts/inode.c 2008-03-24 20:08:57.000000000 -0700
> @@ -201,8 +201,11 @@ static int devpts_get_sb(struct file_sys
> if (IS_ERR(sb))
> return PTR_ERR(sb);
>
> - if (sb->s_root)
> + if (sb->s_root) {
> + if (!(flags & MS_KERNMOUNT))
> + ns->user_mounted = 1;
> return simple_set_mnt(mnt, sb);
> + }
>
> sb->s_flags = flags;
> err = devpts_fill_super(sb, data, flags & MS_SILENT ? 1 : 0);
> @@ -248,6 +251,10 @@ int devpts_new_index(struct pts_namespac
> int index;
> int idr_ret;
>
> + if (!pts_ns || !pts_ns->user_mounted) {
> + printk(KERN_ERR "devpts_new_index() without user_mount\n");
> + return -ENOSYS;

```

```

> + }
> retry:
> if (!idr_pre_get(&pts_ns->allocated_ptys, GFP_KERNEL)) {
> return -ENOMEM;
> @@ -273,7 +280,7 @@ retry:
>
> void devpts_kill_index(struct pts_namespace *pts_ns, int idx)
> {
> -
> + BUG_ON(!pts_ns->user_mounted);
> down(&allocated_ptys_lock);
> idr_remove(&pts_ns->allocated_ptys, idx);
> up(&allocated_ptys_lock);
> @@ -293,6 +300,11 @@ int devpts_pty_new( struct pts_namespace
> BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
> BUG_ON(driver->subtype != PTY_TYPE_SLAVE);
>
> + if (!pts_ns || !pts_ns->user_mounted) {
> + printk(KERN_ERR "devpts_pty_new() without user_mount\n");
> + return -ENOSYS;
> + }
> +
> mnt = pts_ns->mnt;
> root = mnt->mnt_root;
>
> @@ -332,6 +344,11 @@ struct tty_struct *devpts_get_tty(struct
> struct dentry *dentry;
> struct tty_struct *tty;
>
> + if (!pts_ns || !pts_ns->user_mounted) {
> + printk(KERN_ERR "devpts_get_tty() without user_mount\n");
> + return ERR_PTR(-ENOSYS);
> + }
> +
> mnt = pts_ns->mnt;
>
> dentry = get_node(mnt->mnt_root, number);
> @@ -353,6 +370,10 @@ void devpts_pty_kill(struct pts_namespac
> struct dentry *dentry;
> struct dentry *root;
>
> + if (!pts_ns || !pts_ns->user_mounted) {
> + printk(KERN_ERR "devpts_pty_kill() without user_mount\n");
> + BUG_ON(1);
> + }
> root = pts_ns->mnt->mnt_root;
>
> dentry = get_node(root, number);

```

---

Subject: Re: [PATCH 4/7] Implement get\_pts\_ns() and put\_pts\_ns()  
Posted by [serue](#) on Tue, 25 Mar 2008 15:06:07 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):

```
>
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [PATCH 4/7]: Implement get_pts_ns() and put_pts_ns()
>
> Implement get_pts_ns() and put_pts_ns() interfaces.
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> include/linux/devpts_fs.h | 21 ++++++
> 1 file changed, 20 insertions(+), 1 deletion(-)
>
> Index: 2.6.25-rc5-mm1/include/linux/devpts_fs.h
> =====
> --- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:04:31.000000000 -0700
> +++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:05:05.000000000 -0700
> @@ -27,13 +27,26 @@ struct pts_namespace {
> extern struct pts_namespace init_pts_ns;
>
> #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 */
>
> +static inline void free_pts_ns(struct kref *ns_kref) { }
> +
> +static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
> +{
> + if (ns && (ns != &init_pts_ns))
```

How come you need to this extra check? No other namespaces special-case the initial ns this way, do they?

```
> + kref_get(&ns->kref);
> + return ns;
```

```

> +}
> +static inline void put_pts_ns(struct pts_namespace *ns)
> +{
> + if (ns && (ns != &init_pts_ns))
> + kref_put(&ns->kref, free_pts_ns);
> +}
> +
> #else
>
> /* Dummy stubs in the no-pty case */
> @@ -43,6 +56,12 @@ static inline int devpts_pty_new(struct
> static inline struct tty_struct *devpts_get_tty(int number) { return NULL; }
> static inline void devpts_pty_kill(int number) { }
>
> +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) { }
> #endif
>

```

---

Containers mailing list

Containers@lists.linux-foundation.org

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

---



---

Subject: Re: [PATCH 5/7]: Determine pts\_ns from a pty's inode.

Posted by [serue](#) on Tue, 25 Mar 2008 15:17:05 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):

```

>
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [PATCH 5/7]: Determine pts_ns from a pty's inode.
>
> The devpts interfaces currently operate on a specific pts namespace
> which they get from the 'current' task.
>
> With implementation of containers and cloning of PTS namespaces, we want
> to be able to access PTYs in a child-pts-ns from a parent-pts-ns. For
> instance we could bind-mount and pivot-root the child container on
> '/vserver/vserver1' and then access the "pts/0" of 'vserver1' using
>
> $ echo foo > /vserver/vserver1/dev/pts/0
>
> The task doing the above 'echo' could be in parent-pts-ns. So we find

```



> the 'pts-ns' of the above file from the inode representing the above  
 > file rather than from the 'current' task.  
 >  
 > Note that we need to find and hold a reference to the pts\_ns to prevent  
 > the pts\_ns from being freed while it is being accessed from 'outside'.  
 >  
 > This patch implements, 'pts\_ns\_from\_inode()' which returns the pts\_ns  
 > using 'inode->i\_sb->s\_fs\_info'.  
 >  
 > Since, the 'inode' information is not visible inside devpts code itself,  
 > this patch modifies the tty driver code to determine the pts\_ns and passes  
 > it into devpts.  
 >  
 > TODO:  
 > What is the expected behavior when '/dev/tty' or '/dev/ptmx' are  
 > accessed from parent-pts-ns. i.e:  
 >  
 > \$ echo "foobar" > /vserver/vserver1/dev/tty)  
 >  
 > This patch currently ignores the '/vserver/vserver1' part (that

The way this is phrased it almost sounds like you're considering using the pathnames to figure out the ptsns to use :).

It's not clear to me what is the sane thing to do.

what you're doing here - have /dev/ptmx and /dev/tty always use current->s ptsns - isn't ideal.

It would be nicer to not have a 'devpts ns', and instead have a full device namespace. However, then it still isn't clear how to tie /vs/vs1/dev/ptmx to vs1's device namespace, since there is no device fs to which to tie the devns.

We could tie the devns to a device inode on mknod, using the devns of the creating task. Then when starting up vs1, you just have to always let vs1 create /dev/ptmx and /dev/tty. I can't think of anything better offhand.

Other ideas?

Or do we just keep what Suka has?

> seemed to be the easiest to do :-). So opening /dev/ptmx from  
 > even the child pts-ns will create a pty in the \_PARENT\_ pts-ns.

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

```

> drivers/char/pty.c      | 2 -
> drivers/char/tty_io.c   | 86 ++++++-----
> fs/devpts/inode.c       | 19 +++-----
> include/linux/devpts_fs.h | 42 ++++++-----
> 4 files changed, 119 insertions(+), 30 deletions(-)
>
> Index: 2.6.25-rc5-mm1/include/linux/devpts_fs.h
> =====
> --- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:05:05.000000000 -0700
> +++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:08:33.000000000 -0700
> @@ -17,6 +17,7 @@
> #include <linux/nsproxy.h>
> #include <linux/kref.h>
> #include <linux/idr.h>
> +#include <linux/fs.h>
>
> struct pts_namespace {
>     struct kref kref;
> @@ -26,12 +27,43 @@ struct pts_namespace {
>
> extern struct pts_namespace init_pts_ns;
>
> +#define DEVPTS_SUPER_MAGIC 0x1cd1
> +static inline struct pts_namespace *pts_ns_from_inode(struct inode *inode)
> +{
> + /*
> + * Need this bug-on for now to catch any cases in tty_open()
> + * or release_dev() I may have missed.
> + */
> + BUG_ON(inode->i_sb->s_magic != DEVPTS_SUPER_MAGIC);
> +
> + /*
> + * If we have a valid inode, we already have a reference to
> + * mount-point. Since there is a single super-block for the
> + * devpts mount, i_sb->s_fs_info cannot go to NULL. So we
> + * should not need a lock here.
> + */
> +
> + return (struct pts_namespace *)inode->i_sb->s_fs_info;
> +}
> +
> +static inline struct pts_namespace *current_pts_ns(void)
> +{
> + return &init_pts_ns;
> +}
> +
> +
> #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 */
> +int devpts_new_index(struct pts_namespace *pts_ns);
> +void devpts_kill_index(struct pts_namespace *pts_ns, int idx);
> +
> +/* mknod in devpts */
> +int devpts_pty_new(struct pts_namespace *pts_ns, struct tty_struct *tty);
> +
> +/* get tty structure */
> +struct tty_struct *devpts_get_tty(struct pts_namespace *pts_ns, int number);
> +
> +/* unlink */
> +void devpts_pty_kill(struct pts_namespace *pts_ns, int number);
>
> static inline void free_pts_ns(struct kref *ns_kref) { }
>
> Index: 2.6.25-rc5-mm1/drivers/char/tty_io.c
> =====
> --- 2.6.25-rc5-mm1.orig/drivers/char/tty_io.c 2008-03-24 20:04:26.000000000 -0700
> +++ 2.6.25-rc5-mm1/drivers/char/tty_io.c 2008-03-24 20:08:15.000000000 -0700
> @@ -2064,8 +2064,8 @@ static void tty_line_name(struct tty_dri
>  * relaxed for the (most common) case of reopening a tty.
>  */
>
> -static int init_dev(struct tty_driver *driver, int idx,
> - struct tty_struct **ret_tty)
> +static int init_dev(struct tty_driver *driver, struct pts_namespace *pts_ns,
> + int idx, struct tty_struct **ret_tty)
> {
>  struct tty_struct *tty, *o_tty;
>  struct ktermios *tp, **tp_loc, *o_tp, **o_tp_loc;
> @@ -2074,7 +2074,7 @@ static int init_dev(struct tty_driver *d
>
>  /* check whether we're reopening an existing tty */
>  if (driver->flags & TTY_DRIVER_DEVPTS_MEM) {
> - tty = devpts_get_tty(idx);
> + tty = devpts_get_tty(pts_ns, idx);
>  /*
>   * If we don't have a tty here on a slave open, it's because
>   * the master already started the close process and there's
> @@ -2361,6 +2361,43 @@ static void release_tty(struct tty_struc
>  }
>
>  /*
> + * When opening /dev/tty and /dev/ptmx, use the pts-ns of the calling

```

```

> + * process. For any other pts device, use the pts-ns, in which the
> + * device was created. This latter case is needed when the pty is
> + * being accessed from a parent container.
> + *
> + * Eg: Suppose the user used bind-mount and pivot-root to mount a
> + * child- container's root on /vs/vs1. Then "/vs/vs1/dev/pts/0"
> + * in parent container and "/dev/pts/0" in child container would
> + * refer to the same device.
> + *
> + * When parent-container opens, "/vs/vs1/dev/pts/0" we find and
> + * grab/drop reference to child container's pts-ns (using @filp).
> + */
> + struct pts_namespace *pty_pts_ns(struct tty_driver *driver, struct inode *inode)
> + {
> +
> + int devpts;
> + int pty_master;
> + dev_t dev;
> + struct pts_namespace *pts_ns;
> +
> + devpts = (driver->flags & TTY_DRIVER_DEVPTS_MEM) != 0;
> + pty_master = (driver->type == TTY_DRIVER_TYPE_PTY &&
> + driver->subtype == PTY_TYPE_MASTER);
> +
> + pts_ns = NULL;
> + if (devpts) {
> + dev = inode->i_rdev;
> + if (pty_master || dev == MKDEV(TTYAUX_MAJOR, 0))
> + pts_ns = current_pts_ns();
> + else
> + pts_ns = pts_ns_from_inode(inode);
> + }
> + return pts_ns;
> + }
> +
> + /*
> + * Even releasing the tty structures is a tricky business.. We have
> + * to be very careful that the structures are all released at the
> + * same time, as interrupts might otherwise get the wrong pointers.
> + @@ -2376,10 +2413,12 @@ static void release_dev(struct file *fil
> + int idx;
> + char buf[64];
> + unsigned long flags;
> + struct pts_namespace *pts_ns;
> + struct inode *inode;
> +
> + inode = filp->f_path.dentry->d_inode;
> + tty = (struct tty_struct *)filp->private_data;

```

```

> - if (tty_paranoia_check(tty, filp->f_path.dentry->d_inode,
> - "release_dev"))
> + if (tty_paranoia_check(tty, inode, "release_dev"))
> return;
>
> check_tty_count(tty, "release_dev");
> @@ -2391,6 +2430,7 @@ static void release_dev(struct file *fil
>     tty->driver->subtype == PTY_TYPE_MASTER);
> devpts = (tty->driver->flags & TTY_DRIVER_DEVPTS_MEM) != 0;
> o_tty = tty->link;
> + pts_ns = pty_pts_ns(tty->driver, inode);
>
> #ifdef TTY_PARANOIA_CHECK
> if (idx < 0 || idx >= tty->driver->num) {
> @@ -2633,8 +2673,13 @@ static void release_dev(struct file *fil
> release_tty(tty, idx);
>
> /* Make this pty number available for reallocation */
> - if (devpts)
> - devpts_kill_index(idx);
> + if (devpts) {
> + devpts_kill_index(pts_ns, idx);
> + /*
> + * Drop reference got in init_dev()
> + */
> + put_pts_ns(pts_ns);
> + }
> }
>
> /**
> @@ -2666,6 +2711,7 @@ static int tty_open(struct inode *inode,
> int index;
> dev_t device = inode->i_rdev;
> unsigned short saved_flags = filp->f_flags;
> + struct pts_namespace *pts_ns;
>
> nonseekable_open(inode, filp);
>
> @@ -2715,7 +2761,20 @@ retry_open:
> return -ENODEV;
> }
> got_driver:
> - retval = init_dev(driver, index, &tty);
> +
> + /*
> + * What pts-ns do we want to use when opening "/dev/tty" ?
> + * Sounds like current_pts_ns(), but what should happen
> + * if parent pts ns does:

```

```

> + *
> + * echo foo > /vs/vs1/dev/tty
> + *
> + * (See Serge's setupvs1 script for the /vs/vs1...)
> + */
> + pts_ns = pty_pts_ns(driver, inode);
> + get_pts_ns(pts_ns);
> +
> + retval = init_dev(driver, pts_ns, index, &tty);
> mutex_unlock(&tty_mutex);
> if (retval)
> return retval;
@@ -2790,16 +2849,19 @@ static int ptmx_open(struct inode *inode
> struct tty_struct *tty;
> int retval;
> int index;
> + struct pts_namespace *pts_ns;
>
> nonseekable_open(inode, filp);
>
> + pts_ns = current_pts_ns();
> +
> /* find a device that is not in use. */
> - index = devpts_new_index();
> + index = devpts_new_index(pts_ns);
> if (index < 0)
> return index;
>
> mutex_lock(&tty_mutex);
> - retval = init_dev(ptm_driver, index, &tty);
> + retval = init_dev(ptm_driver, pts_ns, index, &tty);
> mutex_unlock(&tty_mutex);
>
> if (retval)
@@ -2809,7 +2871,7 @@ static int ptmx_open(struct inode *inode
> filp->private_data = tty;
> file_move(filp, &tty->tty_files);
>
> - retval = devpts_pty_new(tty->link);
> + retval = devpts_pty_new(pts_ns, tty->link);
> if (retval)
> goto out1;
>
> @@ -2821,7 +2883,7 @@ out1:
> release_dev(filp);
> return retval;
> out:
> - devpts_kill_index(index);

```

```

> + devpts_kill_index(pts_ns, index);
> return retval;
> }
> #endif
> Index: 2.6.25-rc5-mm1/fs/devpts/inode.c
> =====
> --- 2.6.25-rc5-mm1.orig/fs/devpts/inode.c 2008-03-24 20:04:31.000000000 -0700
> +++ 2.6.25-rc5-mm1/fs/devpts/inode.c 2008-03-24 20:08:33.000000000 -0700
> @@ -23,8 +23,6 @@
> #include <linux/fsnotify.h>
> #include <linux/seq_file.h>
>
> -#define DEVPTS_SUPER_MAGIC 0x1cd1
> -
> #define DEVPTS_DEFAULT_MODE 0600
>
> extern int pty_limit; /* Config limit on Unix98 ptys */
> @@ -245,11 +243,10 @@ static struct dentry *get_node(struct de
> return lookup_one_len(s, root, sprintf(s, "%d", num));
> }
>
> -int devpts_new_index(void)
> +int devpts_new_index(struct pts_namespace *pts_ns)
> {
> int index;
> int idr_ret;
> - struct pts_namespace *pts_ns = &init_pts_ns;
>
> retry:
> if (!idr_pre_get(&pts_ns->allocated_ptys, GFP_KERNEL)) {
> @@ -274,16 +271,15 @@ retry:
> return index;
> }
>
> -void devpts_kill_index(int idx)
> +void devpts_kill_index(struct pts_namespace *pts_ns, int idx)
> {
> - struct pts_namespace *pts_ns = &init_pts_ns;
>
> down(&allocated_ptys_lock);
> idr_remove(&pts_ns->allocated_ptys, idx);
> up(&allocated_ptys_lock);
> }
>
> -int devpts_pty_new(struct tty_struct *tty)
> +int devpts_pty_new( struct pts_namespace *pts_ns, struct tty_struct *tty)
> {
> int number = tty->index; /* tty layer puts index from devpts_new_index() in here */

```

```

> struct tty_driver *driver = tty->driver;
> @@ -292,7 +288,6 @@ int devpts_pty_new(struct tty_struct *tt
> struct dentry *root;
> struct vfsmount *mnt;
> struct inode *inode;
> - struct pts_namespace *pts_ns = &init_pts_ns;
>
> /* We're supposed to be given the slave end of a pty */
> BUG_ON(driver->type != TTY_DRIVER_TYPE_PTY);
> @@ -331,13 +326,13 @@ int devpts_pty_new(struct tty_struct *tt
> return 0;
> }
>
> -struct tty_struct *devpts_get_tty(int number)
> +struct tty_struct *devpts_get_tty(struct pts_namespace *pts_ns, int number)
> {
> struct vfsmount *mnt;
> struct dentry *dentry;
> struct tty_struct *tty;
>
> - mnt = init_pts_ns.mnt;
> + mnt = pts_ns->mnt;
>
> dentry = get_node(mnt->mnt_root, number);
>
> @@ -353,12 +348,12 @@ struct tty_struct *devpts_get_tty(int nu
> return tty;
> }
>
> -void devpts_pty_kill(int number)
> +void devpts_pty_kill(struct pts_namespace *pts_ns, int number)
> {
> struct dentry *dentry;
> struct dentry *root;
>
> - root = init_pts_ns.mnt->mnt_root;
> + root = pts_ns->mnt->mnt_root;
>
> dentry = get_node(root, number);
>
> Index: 2.6.25-rc5-mm1/drivers/char/pty.c
> =====
> --- 2.6.25-rc5-mm1.orig/drivers/char/pty.c 2008-03-24 20:02:57.000000000 -0700
> +++ 2.6.25-rc5-mm1/drivers/char/pty.c 2008-03-24 20:08:15.000000000 -0700
> @@ -59,7 +59,7 @@ static void pty_close(struct tty_struct
> set_bit(TTY_OTHER_CLOSED, &tty->flags);
> #ifdef CONFIG_UNIX98_PTYS
> if (tty->driver == ptm_driver)

```



```
> - devpts_pty_kill(tty->index);
> + devpts_pty_kill(current_pts_ns(), tty->index);
> #endif
> tty_vhangup(tty->link);
> }
```

---

Containers mailing list

Containers@lists.linux-foundation.org

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

---

---

Subject: Re: [PATCH 4/7] Implement get\_pts\_ns() and put\_pts\_ns()

Posted by [serue](#) on Tue, 25 Mar 2008 15:29:03 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):

```
>
> From: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> Subject: [PATCH 4/7]: Implement get_pts_ns() and put_pts_ns()
>
> Implement get_pts_ns() and put_pts_ns() interfaces.
>
> Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>
> ---
> include/linux/devpts_fs.h | 21 ++++++
> 1 file changed, 20 insertions(+), 1 deletion(-)
>
> Index: 2.6.25-rc5-mm1/include/linux/devpts_fs.h
> =====
> --- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:04:31.000000000 -0700
> +++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:05:05.000000000 -0700
> @@ -27,13 +27,26 @@ struct pts_namespace {
> extern struct pts_namespace init_pts_ns;
>
> #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 */
>
> +static inline void free_pts_ns(struct kref *ns_kref) { }
> +
> +static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
> +{
> + if (ns && (ns != &init_pts_ns))
> + kref_get(&ns->kref);
```

```

> + return ns;
> +}
> +static inline void put_pts_ns(struct pts_namespace *ns)
> +{
> + if (ns && (ns != &init_pts_ns))
> + kref_put(&ns->kref, free_pts_ns);

```

This isn't right, or I'm not thinking right. Don't you somewhere need to

1. rcu\_assign ns->mnt->mnt\_sb->s\_fs\_info to NULL
2. wait a grace period
3. call free\_pts\_ns and check the refcount on the ns again?

and then do pts\_ns\_from\_inode() under an rcu\_read\_lock and grab a ref to the ns?

```

> +}
> +
> #else
>
> /* Dummy stubs in the no-pty case */
> @@ -43,6 +56,12 @@ static inline int devpts_pty_new(struct
> static inline struct tty_struct *devpts_get_tty(int number) { return NULL; }
> static inline void devpts_pty_kill(int number) { }
>
> +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) { }
> #endif
>

```

---

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

---



---

Subject: Re: [PATCH 6/7]: Check for user-space mount of /dev/pts  
Posted by [Sukadev Bhattiprolu](#) on Tue, 25 Mar 2008 17:25:45 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Serge E. Hallyn [serue@us.ibm.com] wrote:  
| Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):  
| >  
| > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>

| > Subject: [PATCH 6/7]: Check for user-space mount of /dev/pts  
| >  
| > When the pts namespace is cloned, the /dev/pts is not useful unless it  
| > is remounted from the user space.  
| >  
| > If user-space clones pts namespace but does not remount /dev/pts, it  
| > would end up using the /dev/pts mount from parent-pts-ns but allocate  
| > the pts indices from current pts ns.  
|  
| So why not use the allocated\_ptys from the parent ptsns? It's what  
| userspace asked for and it's safe to do.

The problem is when opening /dev/ptmx, we use current\_pts\_ns() and  
when opening slave-pty, we use pts\_ns from the inode.

If child-pts-ns opens /dev/ptmx, we use 'allocated-ptys' from  
child-pts-ns and we allocate index 0. But when the process opens  
the slave pty "/dev/pts/0", we would get the pts\_ns from the  
inode which would come from parent-pts-ns (and could refer to  
an existing pty).

Agree with Alexey and Pavel, its bad. Will think some more, but  
appreciate any ideas.

Sukadev

---

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

---

---

Subject: Re: [PATCH 4/7] Implement get\_pts\_ns() and put\_pts\_ns()  
Posted by [Sukadev Bhattiprolu](#) on Tue, 25 Mar 2008 18:44:16 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Serge E. Hallyn [serue@us.ibm.com] wrote:  
| Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):  
| >  
| > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
| > Subject: [PATCH 4/7]: Implement get\_pts\_ns() and put\_pts\_ns()  
| >  
| > Implement get\_pts\_ns() and put\_pts\_ns() interfaces.  
| >  
| > Signed-off-by: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
| > ---  
| > include/linux/devpts\_fs.h | 21 ++++++++  
| > 1 file changed, 20 insertions(+), 1 deletion(-)  
| >

```

| > Index: 2.6.25-rc5-mm1/include/linux/devpts_fs.h
| > =====
| > --- 2.6.25-rc5-mm1.orig/include/linux/devpts_fs.h 2008-03-24 20:04:31.000000000 -0700
| > +++ 2.6.25-rc5-mm1/include/linux/devpts_fs.h 2008-03-24 20:05:05.000000000 -0700
| > @@ -27,13 +27,26 @@ struct pts_namespace {
| > extern struct pts_namespace init_pts_ns;
| >
| > #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 */
| >
| > +static inline void free_pts_ns(struct kref *ns_kref) { }
| > +
| > +static inline struct pts_namespace *get_pts_ns(struct pts_namespace *ns)
| > +{
| > + if (ns && (ns != &init_pts_ns))
| > + kref_get(&ns->kref);
| > + return ns;
| > +}
| > +static inline void put_pts_ns(struct pts_namespace *ns)
| > +{
| > + if (ns && (ns != &init_pts_ns))
| > + kref_put(&ns->kref, free_pts_ns);

```

| This isn't right, or I'm not thinking right. Don't you somewhere need to

- | 1. rcu\_assign ns->mnt->mnt\_sb->s\_fs\_info to NULL
- | 2. wait a grace period
- | 3. call free\_pts\_ns and check the refcount on the ns again?

| and then do pts\_ns\_from\_inode() under an rcu\_read\_lock and grab a ref to the ns?

Yes, we need the rcu to grab the reference to pts\_ns.

---

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

---



---

Subject: Re: [PATCH 5/7]: Determine pts\_ns from a pty's inode.  
Posted by [serue](#) on Tue, 25 Mar 2008 21:14:06 GMT

Quoting Serge E. Hallyn (serue@us.ibm.com):

> Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):

> >

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

> > Subject: [PATCH 5/7]: Determine pts\_ns from a pty's inode.

> >

> > The devpts interfaces currently operate on a specific pts namespace

> > which they get from the 'current' task.

> >

> > With implementation of containers and cloning of PTS namespaces, we want

> > to be able to access PTYs in a child-pts-ns from a parent-pts-ns. For

> > instance we could bind-mount and pivot-root the child container on

> > '/vserver/vserver1' and then access the "pts/0" of 'vserver1' using

> >

> > \$ echo foo > /vserver/vserver1/dev/pts/0

> >

> > The task doing the above 'echo' could be in parent-pts-ns. So we find

> > the 'pts-ns' of the above file from the inode representing the above

> > file rather than from the 'current' task.

> >

> > Note that we need to find and hold a reference to the pts\_ns to prevent

> > the pts\_ns from being freed while it is being accessed from 'outside'.

> >

> > This patch implements, 'pts\_ns\_from\_inode()' which returns the pts\_ns

> > using 'inode->i\_sb->s\_fs\_info'.

> >

> > Since, the 'inode' information is not visible inside devpts code itself,

> > this patch modifies the tty driver code to determine the pts\_ns and passes

> > it into devpts.

> >

> > TODO:

> > What is the expected behavior when '/dev/tty' or '/dev/ptmx' are

> > accessed from parent-pts-ns. i.e:

> >

> > \$ echo "foobar" > /vserver/vserver1/dev/tty)

> >

> > This patch currently ignores the '/vserver/vserver1' part (that

>

> The way this is phrased it almost sounds like you're considering using

> the pathnames to figure out the ptsns to use :).

>

> It's not clear to me what is the sane thing to do.

>

> what you're doing here - have /dev/ptmx and /dev/tty always use

> current->s ptsns - isn't ideal.

>

> It would be nicer to not have a 'devpts ns', and instead have a

> full device namespace. However, then it still isn't clear how to tie  
> /vs/vs1/dev/ptmx to vs1's device namespace, since there is no device  
> fs to which to tie the devns.  
>  
> We could tie the devns to a device inode on mknod, using the devns of  
> the creating task. Then when starting up vs1, you just have to always  
> let vs1 create /dev/ptmx and /dev/tty. I can't think of anything  
> better offhand.  
>  
> Other ideas?

I suppose you could just create /dev/pts/ptmx and /dev/pts/tty.  
Recommend that in containers /dev/ptmx and /dev/tty be symlinks  
into /dev/pts. Applications don't need to change. If  
ptmx\_open() sees that inode->i\_sb is a devptsfs, it gets the  
namespace from the sb. If not, then it was a device in /dev  
and it gets the namespace from current.

-serge

---

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

---

---

Subject: Re: [PATCH 5/7]: Determine pts\_ns from a pty's inode.  
Posted by [Sukadev Bhattiprolu](#) on Wed, 26 Mar 2008 02:03:28 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Serge E. Hallyn [serue@us.ibm.com] wrote:  
| Quoting Serge E. Hallyn (serue@us.ibm.com):  
| > Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):  
| > >  
| > > From: Sukadev Bhattiprolu <sukadev@us.ibm.com>  
| > > Subject: [PATCH 5/7]: Determine pts\_ns from a pty's inode.  
| > >  
| > > The devpts interfaces currently operate on a specific pts namespace  
| > > which they get from the 'current' task.  
| > >  
| > > With implementation of containers and cloning of PTS namespaces, we want  
| > > to be able to access PTYs in a child-pts-ns from a parent-pts-ns. For  
| > > instance we could bind-mount and pivot-root the child container on  
| > > '/vserver/vserver1' and then access the "pts/0" of 'vserver1' using  
| > >  
| > > \$ echo foo > /vserver/vserver1/dev/pts/0  
| > >  
| > > The task doing the above 'echo' could be in parent-pts-ns. So we find  
| > > the 'pts-ns' of the above file from the inode representing the above

```

| > > file rather than from the 'current' task.
| > >
| > > Note that we need to find and hold a reference to the pts_ns to prevent
| > > the pts_ns from being freed while it is being accessed from 'outside'.
| > >
| > > This patch implements, 'pts_ns_from_inode()' which returns the pts_ns
| > > using 'inode->i_sb->s_fs_info'.
| > >
| > > Since, the 'inode' information is not visible inside devpts code itself,
| > > this patch modifies the tty driver code to determine the pts_ns and passes
| > > it into devpts.
| > >
| > > TODO:
| > > What is the expected behavior when '/dev/tty' or '/dev/ptmx' are
| > > accessed from parent-pts-ns. i.e:
| > >
| > > $ echo "foobar" > /vserver/vserver1/dev/tty)
| > >
| > > This patch currently ignores the '/vserver/vserver1' part (that
| >
| > The way this is phrased it almost sounds like you're considering using
| > the pathnames to figure out the ptsns to use :).
| >
| > It's not clear to me what is the sane thing to do.
| >
| > what you're doing here - have /dev/ptmx and /dev/tty always use
| > current->'s ptsns - isn't ideal.
| >
| > It would be nicer to not have a 'devpts ns', and instead have a
| > full device namespace. However, then it still isn't clear how to tie
| > /vs/vs1/dev/ptmx to vs1's device namespace, since there is no device
| > fs to which to tie the devns.
| >
| > We could tie the devns to a device inode on mknod, using the devns of
| > the creating task. Then when starting up vs1, you just have to always
| > let vs1 create /dev/ptmx and /dev/tty. I can't think of anything
| > better offhand.
| >
| > Other ideas?
|
| I suppose you could just create /dev/pts/ptmx and /dev/pts/tty.
| Recommend that in containers /dev/ptmx and /dev/tty be symlinks
| into /dev/pts. Applications don't need to change. If
| ptmx_open() sees that inode->i_sb is a devptsfs, it gets the
| namespace from the sb. If not, then it was a device in /dev
| and it gets the nmespace from current.

```

But we would still depend on user-space remounting /dev/pts after

the clone right ? Until they do that we would access the parent container's /dev/pts/ptmx ?

---

Containers mailing list

Containers@lists.linux-foundation.org

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

---

---

Subject: Re: [PATCH 5/7]: Determine pts\_ns from a pty's inode.

Posted by [serue](#) on Wed, 26 Mar 2008 02:50:38 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Quoting [sukadev@us.ibm.com](mailto:sukadev@us.ibm.com) ([sukadev@us.ibm.com](mailto:sukadev@us.ibm.com)):

> Serge E. Hallyn [[serue@us.ibm.com](mailto:serue@us.ibm.com)] wrote:

> | Quoting Serge E. Hallyn ([serue@us.ibm.com](mailto:serue@us.ibm.com)):

> | > Quoting [sukadev@us.ibm.com](mailto:sukadev@us.ibm.com) ([sukadev@us.ibm.com](mailto:sukadev@us.ibm.com)):

> | > >

> | > > From: Sukadev Bhattiprolu <[sukadev@us.ibm.com](mailto:sukadev@us.ibm.com)>

> | > > Subject: [PATCH 5/7]: Determine pts\_ns from a pty's inode.

> | > >

> | > > The devpts interfaces currently operate on a specific pts namespace

> | > > which they get from the 'current' task.

> | > >

> | > > With implementation of containers and cloning of PTS namespaces, we want

> | > > to be able to access PTYs in a child-pts-ns from a parent-pts-ns. For

> | > > instance we could bind-mount and pivot-root the child container on

> | > > '/vserver/vserver1' and then access the "pts/0" of 'vserver1' using

> | > >

> | > > \$ echo foo > /vserver/vserver1/dev/pts/0

> | > >

> | > > The task doing the above 'echo' could be in parent-pts-ns. So we find

> | > > the 'pts-ns' of the above file from the inode representing the above

> | > > file rather than from the 'current' task.

> | > >

> | > > Note that we need to find and hold a reference to the pts\_ns to prevent

> | > > the pts\_ns from being freed while it is being accessed from 'outside'.

> | > >

> | > > This patch implements, 'pts\_ns\_from\_inode()' which returns the pts\_ns

> | > > using 'inode->i\_sb->s\_fs\_info'.

> | > >

> | > > Since, the 'inode' information is not visible inside devpts code itself,

> | > > this patch modifies the tty driver code to determine the pts\_ns and passes

> | > > it into devpts.

> | > >

> | > > TODO:

> | > > What is the expected behavior when '/dev/tty' or '/dev/ptmx' are

> | > > accessed from parent-pts-ns. i.e:

> | > >



```

> | > > $ echo "foobar" > /vserver/vserver1/dev/tty)
> | > >
> | > > This patch currently ignores the '/vserver/vserver1' part (that
> | >
> | > The way this is phrased it almost sounds like you're considering using
> | > the pathnames to figure out the ptsns to use :).
> | >
> | > It's not clear to me what is the sane thing to do.
> | >
> | > what you're doing here - have /dev/ptmx and /dev/tty always use
> | > current->'s ptsns - isn't ideal.
> | >
> | > It would be nicer to not have a 'devpts ns', and instead have a
> | > full device namespace. However, then it still isn't clear how to tie
> | > /vs/vs1/dev/ptmx to vs1's device namespace, since there is no device
> | > fs to which to tie the devns.
> | >
> | > We could tie the devns to a device inode on mknod, using the devns of
> | > the creating task. Then when starting up vs1, you just have to always
> | > let vs1 create /dev/ptmx and /dev/tty. I can't think of anything
> | > better offhand.
> | >
> | > Other ideas?
> |
> | I suppose you could just create /dev/pts/ptmx and /dev/pts/tty.
> | Recommend that in containers /dev/ptmx and /dev/tty be symlinks
> | into /dev/pts. Applications don't need to change. If
> | ptmx_open() sees that inode->i_sb is a devptsfs, it gets the
> | namespace from the sb. If not, then it was a device in /dev
> | and it gets the namespace from current.
>
> But we would still depend on user-space remounting /dev/pts after
> the clone right ? Until they do that we would access the parent
> container's /dev/pts/ptmx ?

```

Yes. Which is the right thing to do imo.

-serge

---

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

---

Subject: Re: [PATCH 5/7]: Determine pts\_ns from a pty's inode.  
Posted by [Sukadev Bhattiprolu](#) on Wed, 26 Mar 2008 14:55:21 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Serge E. Hallyn [serue@us.ibm.com] wrote:

```
| > | I suppose you could just create /dev/pts/ptmx and /dev/pts/tty.  
| > | Recommend that in containers /dev/ptmx and /dev/tty be symlinks  
| > | into /dev/pts. Applications don't need to change. If  
| > | ptmx_open() sees that inode->i_sb is a devptsfs, it gets the  
| > | namespace from the sb. If not, then it was a device in /dev  
| > | and it gets the namespace from current.  
| >  
| > But we would still depend on user-space remounting /dev/pts after  
| > the clone right ? Until they do that we would access the parent  
| > container's /dev/pts/ptmx ?  
|  
| Yes. Which is the right thing to do imo.
```

Hmm, that sounds reasonable, although slightly inconsistent with pid-ns, where pid starts at 1 regardless of whether /proc is remounted.

But even so, if user fails to establish the symlink, clones the pts ns and tries to create a pty, we would end up with different pts nses again ?

i.e

```
/dev/ptmx is still a char dev in root fs  
clone(pts_ns)  
( In child, (before remount /dev/pts))  
open("/dev/ptmx")  
open("/dev/pts/0")
```

Since ptmx is not in devpts, we use current\_pts\_ns() or child-pts-ns  
Since /dev/pts is not remounted in child, we get the parent pts-ns from

If we can somehow detect the incorrect configuration and fail either  
open, we should be ok :-)  
inode.

Suka

---

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

---

---

Subject: Re: [PATCH 5/7]: Determine pts\_ns from a pty's inode.  
Posted by [serue](#) on Wed, 26 Mar 2008 15:12:05 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):  
> Serge E. Hallyn [serue@us.ibm.com] wrote:  
> | > | I suppose you could just create /dev/pts/ptmx and /dev/pts/tty.

> | > | Recommend that in containers /dev/ptmx and /dev/tty be symlinks  
 > | > | into /dev/pts. Applications don't need to change. If  
 > | > | ptmx\_open() sees that inode->i\_sb is a devptsfs, it gets the  
 > | > | namespace from the sb. If not, then it was a device in /dev  
 > | > | and it gets the namespace from current.  
 > | >  
 > | > But we would still depend on user-space remounting /dev/pts after  
 > | > the clone right ? Until they do that we would access the parent  
 > | > container's /dev/pts/ptmx ?  
 > |  
 > | Yes. Which is the right thing to do imo.  
 >  
 > Hmm, that sounds reasonable, although slightly inconsistent with pid-ns,  
 > where pid starts at 1 regardless of whether /proc is remounted.

Very different cases. The pid is the task's pid in the new pidns.  
 The task ALSO has a different pid in the parent pidns.

The pts only has an identity in one ptsns.

> But even so, if user fails to establish the symlink, clones the pts ns  
 > and tries to create a pty, we would end up with different pts nses again ?

Yes. So what?

> i.e  
 > /dev/ptmx is still a char dev in root fs  
 > clone(pts\_ns)  
 > ( In child, (before remount /dev/pts))  
 > open("/dev/ptmx")  
 > open("/dev/pts/0")  
 >  
 > Since ptmx is not in devpts, we use current\_pts\_ns() or child-pts-ns  
 > Since /dev/pts is not remounted in child, we get the parent pts-ns from  
 >  
 > If we can somehow detect the incorrect configuration and fail either  
 > open, we should be ok :-)

I completely disagree with this sentiment. The kernel doesn't need  
 to detect an "incorrect configuration" if it isn't dangerous. One  
 man's "incorrect configuration" is another man's useful trick.

-serge

---

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

---

Subject: Re: [PATCH 5/7]: Determine pts\_ns from a pty's inode.  
Posted by [Sukadev Bhattiprolu](#) on Wed, 26 Mar 2008 15:18:43 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Serge E. Hallyn [serue@us.ibm.com] wrote:

| Quoting sukadev@us.ibm.com (sukadev@us.ibm.com):

| > Serge E. Hallyn [serue@us.ibm.com] wrote:

| > | > | I suppose you could just create /dev/pts/ptmx and /dev/pts/tty.

| > | > | Recommend that in containers /dev/ptmx and /dev/tty be symlinks

| > | > | into /dev/pts. Applications don't need to change. If

| > | > | ptmx\_open() sees that inode->i\_sb is a devptsfs, it gets the

| > | > | namespace from the sb. If not, then it was a device in /dev

| > | > | and it gets the namespace from current.

| > | >

| > | > But we would still depend on user-space remounting /dev/pts after

| > | > the clone right ? Until they do that we would access the parent

| > | > container's /dev/pts/ptmx ?

| > |

| > | Yes. Which is the right thing to do imo.

| >

| > Hmm, that sounds reasonable, although slightly inconsistent with pid-ns,

| > where pid starts at 1 regardless of whether /proc is remounted.

| Very different cases. The pid is the task's pid in the new pidns.

| The task ALSO has a different pid in the parent pidns.

| The pts only has an identity in one ptsns.

| > But even so, if user fails to establish the symlink, clones the pts ns

| > and tries to create a pty, we would end up with different pts nses again ?

| Yes. So what?

We would end up allocating a pts index from child-pts-ns (i.e index 0)  
and attempt to open /dev/pts/0 which could be an existing pty in the  
parent pts ns ?

| > i.e

| > /dev/ptmx is still a char dev in root fs

| > clone(pts\_ns)

| > ( In child, (before remount /dev/pts))

| > open("/dev/ptmx")

| > open("/dev/pts/0")

| >

| > Since ptmx is not in devpts, we use current\_pts\_ns() or child-pts-ns

| > Since /dev/pts is not remounted in child, we get the parent pts-ns from

| >

| > If we can somehow detect the incorrect configuration and fail either

| > open, we should be ok :-)

|  
| I completely disagree with this sentiment. The kernel doesn't need  
| to detect an "incorrect configuration" if it isn't dangerous. One  
| man's "incorrect configuration" is another man's useful trick.

Myabe configuration is the wrong word, but unless I am missing something  
above, spanning two pts-nses is an error condition ?

---

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

---

---

Subject: Re: [PATCH 5/7]: Determine pts\_ns from a pty's inode.  
Posted by [serue](#) on Wed, 26 Mar 2008 15:43:44 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

Quoting [sukadev@us.ibm.com](mailto:sukadev@us.ibm.com) ([sukadev@us.ibm.com](mailto:sukadev@us.ibm.com)):  
> Serge E. Hallyn [[serue@us.ibm.com](mailto:serue@us.ibm.com)] wrote:  
> | Quoting [sukadev@us.ibm.com](mailto:sukadev@us.ibm.com) ([sukadev@us.ibm.com](mailto:sukadev@us.ibm.com)):  
> | > Serge E. Hallyn [[serue@us.ibm.com](mailto:serue@us.ibm.com)] wrote:  
> | > > | I suppose you could just create /dev/pts/ptmx and /dev/pts/tty.  
> | > > | Recommend that in containers /dev/ptmx and /dev/tty be symlinks  
> | > > | into /dev/pts. Applications don't need to change. If  
> | > > | ptmx\_open() sees that inode->i\_sb is a devptsfs, it gets the  
> | > > | namespace from the sb. If not, then it was a device in /dev  
> | > > | and it gets the nmespace from current.  
> | > >  
> | > > But we would still depend on user-space remounting /dev/pts after  
> | > > the clone right ? Until they do that we would access the parent  
> | > > container's /dev/pts/ptmx ?  
> | > |  
> | > | Yes. Which is the right thing to do imo.  
> | >  
> | > Hmm, that sounds reasonable, although slightly inconsistent with pid-ns,  
> | > where pid starts at 1 regardless of whether /proc is remounted.  
> |  
> | Very different cases. The pid is the task's pid in the new pidns.  
> | The task ALSO has a different pid in the parent pidns.  
> |  
> | The pts only has an identity in one ptsns.  
> |  
> | > But even so, if user fails to establish the symlink, clones the pts ns  
> | > and tries to create a pty, we would end up with different pts nses again ?  
> |  
> | Yes. So what?  
>  
> We would end up allocating a pts index from child-pts-ns (i.e index 0)

> and attempt to open /dev/pts/0 which could be an existing pty in the  
> parent pts ns ?

An SELinux policy tagging child devpts entries with vps1\_u:vps1\_r:vps1\_pts\_t  
and not allowing vps1\_t access to host\_pts\_t entries would forbid it if  
you wanted. But failing that, the kernel doesn't break, so I don't  
it's a problem.

> | > i.e  
> | > /dev/ptmx is still a char dev in root fs  
> | > clone(pts\_ns)  
> | > ( In child, (before remount /dev/pts))  
> | > open("/dev/ptmx")  
> | > open("/dev/pts/0")  
> | >  
> | > Since ptmx is not in devpts, we use current\_pts\_ns() or child-pts-ns  
> | > Since /dev/pts is not remounted in child, we get the parent pts-ns from  
> | >  
> | > If we can somehow detect the incorrect configuration and fail either  
> | > open, we should be ok :-)  
> |  
> | I completely disagree with this sentiment. The kernel doesn't need  
> | to detect an "incorrect configuration" if it isn't dangerous. One  
> | man's "incorrect configuration" is another man's useful trick.  
>  
> Maybe configuration is the wrong word, but unless I am missing something  
> above, spanning two pts-nses is an error condition ?

For userspace, but it doesn't crash the kernel. Userspace didn't set  
things up right, so it gets the wrong thing. If I do a dup2 into fd 3  
and then try to read from fd 4, I get the wrong data. Is that the  
kernel's fault?

-serge

---

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

---