
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>
