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