Subject: Re: [patch 1/8] add user mounts to the kernel
Posted by ebiederm on Sat, 21 Apr 2007 13:14:23 GMT
View Forum Message <> Reply to Message

Miklos Szeredi <miklos@szeredi.hu> writes:

> From: Miklos Szeredi <mszeredi@suse.cz>
>
> Add ownership information to mounts.
>
> A new mount flag, MS_SETUSER is used to make a mount owned by a user.
> If this flag is specified, then the owner will be set to the current
> real user id and the mount will be marked with the MNT_USER flag. On
> remount don't preserve previous owner, and treat MS_SETUSER as for a
> new mount. The MS_SETUSER flag is ignored on mount move.
>
> The MNT_USER flag is not copied on any kind of mount cloning:
> namespace creation, binding or propagation.

I half agree, and as an initial approximation this works.
Ultimately we should be at the point that for mount propagation
that we copy the owner of the from the owner of our parent mount
at the propagation destination.

Mount propagation semantics are a major pain.

> For bind mounts the
> cloned mount(s) are set to MNT_USER depending on the MS_SETUSER mount
> flag. In all the other cases MNT_USER is always cleared.
>
> For MNT_USER mounts a "user=UID" option is added to /proc/PID/mounts.
> This is compatible with how mount ownership is stored in /etc/mtab.

Ok. While I generally agree with the concept this can be simplified some
more.

Eric

> Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>
> ---
>
> Index: linux/fs/namespace.c
> ===
> --- linux.orig/fs/namespace.c 2007-04-20 11:55:02.000000000 +0200
> +++ linux/fs/namespace.c 2007-04-20 11:55:05.000000000 +0200
> @@ -227,6 +227,13 @@ static struct vfsmount *skip_mnt_tree(st
>   return p;

> }
>
> +static void set_mnt_user(struct vfsmount *mnt)
> +{
> + BUG_ON(mnt->mnt_flags & MNT_USER);
> + mnt->mnt_uid = current->uid;

This should be based on fsuid. Unless I'm completely confused.

I think getting the mount user from current when we do mount
propagation could be a problem. In particular I'm pretty
certain in the mount propagation case the mount should be owned
by the user who owns the destination mount that is above us.

> + mnt->mnt_flags |= MNT_USER;
> +}
> +
> static struct vfsmount *clone_mnt(struct vfsmount *old, struct dentry *root,
> int flag)
> {
> @@ -241,6 +248,11 @@ static struct vfsmount *clone_mnt(struct
> mnt->mnt_mountpoint = mnt->mnt_root;
> mnt->mnt_parent = mnt;
>
> + /* don't copy the MNT_USER flag */
> + mnt->mnt_flags &= ~MNT_USER;
> + if (flag & CL_SETUSER)
> + set_mnt_user(mnt);
> +
> if (flag & CL_SLAVE) {
> list_add(&mnt->mnt_slave, &old->mnt_slave_list);
> mnt->mnt_master = old;
> @@ -403,6 +415,8 @@ static int show_vfsmnt(struct seq_file *
> if (mnt->mnt_flags & fs_infop->flag)
> seq_puts(m, fs_infop->str);
> }
> + if (mnt->mnt_flags & MNT_USER)
> + seq_printf(m, ",user=%i", mnt->mnt_uid);
How about making the test "if (mnt->mnt_user != &root_user)"

> if (mnt->mnt_sb->s_op->show_options)
> err = mnt->mnt_sb->s_op->show_options(m, mnt);
> seq_puts(m, " 0 0\n");
> @@ -920,8 +934,9 @@ static int do_change_type(struct nameida
> /*
> * do loopback mount.
> */
> -static int do_loopback(struct nameidata *nd, char *old_name, int recurse)

```
> +static int do_loopback(struct nameidata *nd, char *old_name, int flags)
>  {
> + int clone_flags;
>   struct nameidata old_nd;
>   struct vfsmount *mnt = NULL;
>   int err = mount_is_safe(nd);
> @@ -941,11 +956,12 @@ static int do_loopback(struct nameidata
>   if (!check_mnt(nd->mnt) || !check_mnt(old_nd.mnt))
>    goto out;
>
> + clone_flags = (flags & MS_SETUSER) ? CL_SETUSER : 0;
>   err = -ENOMEM;
> - if (recurse)
> -  mnt = copy_tree(old_nd.mnt, old_nd.dentry, 0);
> + if (flags & MS_REC)
> +  mnt = copy_tree(old_nd.mnt, old_nd.dentry, clone_flags);
>   else
> -  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, 0);
> +  mnt = clone_mnt(old_nd.mnt, old_nd.dentry, clone_flags);
>
>   if (!mnt)
>    goto out;
> @@ -987,8 +1003,11 @@ static int do_remount(struct nameidata *
>
>   down_write(&sb->s_umount);
>   err = do_remount_sb(sb, flags, data, 0);
> - if (!err)
> + if (!err) {
>    nd->mnt->mnt_flags = mnt_flags;
> +  if (flags & MS_SETUSER)
> +   set_mnt_user(nd->mnt);
> + }
>   up_write(&sb->s_umount);
>   if (!err)
>    security_sb_post_remount(nd->mnt, flags, data);
> @@ -1093,10 +1112,13 @@ static int do_new_mount(struct nameidata
>   if (!capable(CAP_SYS_ADMIN))
>    return -EPERM;
>
> - mnt = do_kern_mount(type, flags, name, data);
> + mnt = do_kern_mount(type, flags & ~MS_SETUSER, name, data);
>   if (IS_ERR(mnt))
>    return PTR_ERR(mnt);
>
> + if (flags & MS_SETUSER)
> +  set_mnt_user(mnt);
> +
>   return do_add_mount(mnt, nd, mnt_flags, NULL);
```

> }
>
> @@ -1127,7 +1149,8 @@ int do_add_mount(struct vfsmount *newmnt
> if (S_ISLNK(newmnt->mnt_root->d_inode->i_mode))
> goto unlock;
>
> - newmnt->mnt_flags = mnt_flags;
> + /* MNT_USER was set earlier */
> + newmnt->mnt_flags |= mnt_flags;
> if ((err = graft_tree(newmnt, nd)))
> goto unlock;
>
> @@ -1447,7 +1470,7 @@ long do_mount(char *dev_name, char *dir_
> retval = do_remount(&nd, flags & ~MS_REMOUNT, mnt_flags,
> data_page);
> else if (flags & MS_BIND)
> - retval = do_loopback(&nd, dev_name, flags & MS_REC);
> + retval = do_loopback(&nd, dev_name, flags);
> else if (flags & (MS_SHARED | MS_PRIVATE | MS_SLAVE | MS_UNBINDABLE))
> retval = do_change_type(&nd, flags);
> else if (flags & MS_MOVE)
> Index: linux/include/linux/fs.h
> ===
> --- linux.orig/include/linux/fs.h 2007-04-20 11:55:02.000000000 +0200
> +++ linux/include/linux/fs.h 2007-04-20 11:55:05.000000000 +0200
> @@ -123,6 +123,7 @@ extern int dir_notify_enable;
> #define MS_SLAVE (1<<19) /* change to slave */
> #define MS_SHARED (1<<20) /* change to shared */
> #define MS_RELATIME (1<<21) /* Update atime relative to mtime/ctime. */
> +#define MS_SETUSER (1<<22) /* set mnt_uid to current user */

If we unconditionally use the fsuid I think we can get away without
this flag.

> #define MS_ACTIVE (1<<30)
> #define MS_NOUSER (1<<31)
>
> Index: linux/include/linux/mount.h
> ===
> --- linux.orig/include/linux/mount.h 2007-04-20 11:55:02.000000000 +0200
> +++ linux/include/linux/mount.h 2007-04-20 11:55:05.000000000 +0200
> @@ -30,6 +30,7 @@ struct mnt_namespace;
> #define MNT_RELATIME 0x20
>
> #define MNT_SHRINKABLE 0x100
> +#define MNT_USER 0x200

If we assign a user to all mount points and root gets to own the

initial set of mounts then we don't need the internal MNT_USER
flag.

> #define MNT_SHARED 0x1000 /* if the vfsmount is a shared mount */
> #define MNT_UNBINDABLE 0x2000 /* if the vfsmount is a unbindable mount */
> @@ -61,6 +62,8 @@ struct vfsmount {
>   atomic_t mnt_count;
>   int mnt_expiry_mark; /* true if marked for expiry */
>   int mnt_pinned;
> +
> + uid_t mnt_uid;   /* owner of the mount */

Can we please make this a user struct. That requires a bit of
reference counting but it has uid namespace benefits as well
as making it easy to implement per user mount rlimits.

> };
>
> static inline struct vfsmount *mntget(struct vfsmount *mnt)
> Index: linux/fs/pnode.h
> ===
> --- linux.orig/fs/pnode.h 2007-04-20 11:55:02.000000000 +0200
> +++ linux/fs/pnode.h 2007-04-20 11:55:05.000000000 +0200
> @@ -22,6 +22,7 @@
> #define CL_COPY_ALL   0x04
> #define CL_MAKE_SHARED   0x08
> #define CL_PROPAGATION   0x10
> +#define CL_SETUSER 0x20

> static inline void set_mnt_shared(struct vfsmount *mnt)
> {
>
> --

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