Subject: [patch 00/10] (resend) mount ownership and unprivileged mount syscall
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:41 GMT
View Forum Message <> Reply to Message

This patchset adds support for keeping mount ownership information in
the kernel, and allow unprivileged mount(2) and umount(2) in certain
cases.

This can be useful for the following reasons:

- mount(8) can store ownership ("user=XY" option) in the kernel
  instead, or in addition to storing it in /etc/mtab. For example if
  private namespaces are used with mount propagations /etc/mtab
  becomes unworkable, but using /proc/mounts works fine

- fuse won't need a special suid-root mount/umount utility. Plain
  umount(8) can easily be made to work with unprivileged fuse mounts

- users can use bind mounts without having to pre-configure them in
  /etc/fstab

Unprivileged mounts are restricted to private namespaces created with
a special clone flag.

Changes from the previous submission:

 - add namespace flag for allowing user mounts
 - add clone flag to set above namespace flag
 - make max number of user mounts default to 1024, since now the
   namespace flag will prevent user mounts by default

--

_____

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

Subject: [patch 01/10] add user mounts to the kernel
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:42 GMT
View Forum Message <> Reply to Message

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. 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.

It is expected, that in the future mount(8) will use MS_SETUSER to
store mount ownership within the kernel. This would help in
situations, where /etc/mtab is difficult or impossible to work with,
e.g. when using mount propagation.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/namespace.c
===
--- linux.orig/fs/namespace.c 2007-04-11 18:27:46.000000000 +0200
+++ linux/fs/namespace.c 2007-04-11 20:07:51.000000000 +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;
+ 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);
  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 2007-04-11 18:27:46.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-11 20:07:51.000000000 +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 */
 #define MS_ACTIVE (1<<30)
 #define MS_NOUSER (1<<31)
```

```
Index: linux/include/linux/mount.h
===================================================================
--- linux.orig/include/linux/mount.h 2007-04-11 18:27:33.000000000 +0200
+++ linux/include/linux/mount.h 2007-04-11 20:07:51.000000000 +0200
@@ -28,6 +28,7 @@ struct mnt_namespace;
 #define MNT_NOATIME 0x08
 #define MNT_NODIRATIME 0x10
 #define MNT_RELATIME 0x20
+#define MNT_USER 0x40

 #define MNT_SHRINKABLE 0x100

@@ -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 */
 };

 static inline struct vfsmount *mntget(struct vfsmount *mnt)
Index: linux/fs/pnode.h
===================================================================
--- linux.orig/fs/pnode.h 2007-02-04 19:44:54.000000000 +0100
+++ linux/fs/pnode.h 2007-04-11 20:07:51.000000000 +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
```

Subject: [patch 02/10] allow unprivileged umount
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:43 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

The owner doesn't need sysadmin capabilities to call umount().

Similar behavior as umount(8) on mounts having "user=UID" option in
/etc/mtab. The difference is that umount also checks /etc/fstab,
presumably to exclude another mount on the same mountpoint.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/namespace.c
```
===================================================================
--- linux.orig/fs/namespace.c 2007-04-11 20:07:51.000000000 +0200
+++ linux/fs/namespace.c 2007-04-11 20:08:05.000000000 +0200
@@ -659,6 +659,25 @@ static int do_umount(struct vfsmount *mn
 }

 /*
+ * umount is permitted for
+ *  - sysadmin
+ *  - mount owner, if not forced umount
+ */
+static bool permit_umount(struct vfsmount *mnt, int flags)
+{
+ if (capable(CAP_SYS_ADMIN))
+  return true;
+
+ if (!(mnt->mnt_flags & MNT_USER))
+  return false;
+
+ if (flags & MNT_FORCE)
+  return false;
+
+ return mnt->mnt_uid == current->uid;
+}
+
+/*
  * Now umount can handle mount points as well as block devices.
  * This is important for filesystems which use unnamed block devices.
  *
@@ -681,7 +700,7 @@ asmlinkage long sys_umount(char __user *
   goto dput_and_out;

  retval = -EPERM;
- if (!capable(CAP_SYS_ADMIN))
+ if (!permit_umount(nd.mnt, flags))
   goto dput_and_out;

  retval = do_umount(nd.mnt, flags);
```

--

_____

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

---

Subject: [patch 03/10] account user mounts
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:44 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Add sysctl variables for accounting and limiting the number of user
mounts.

The maximum number of user mounts is set to 1024 by default. This
won't in itself enable user mounts, setting the "permit user mount in
namespace" flag will also be needed.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/include/linux/sysctl.h
===
--- linux.orig/include/linux/sysctl.h 2007-04-11 18:27:46.000000000 +0200
+++ linux/include/linux/sysctl.h 2007-04-11 20:08:16.000000000 +0200
@@ -818,6 +818,8 @@ enum
  FS_AIO_NR=18, /* current system-wide number of aio requests */
  FS_AIO_MAX_NR=19, /* system-wide maximum number of aio requests */
  FS_INOTIFY=20, /* inotify submenu */
+ FS_NR_USER_MOUNTS=21, /* int:current number of user mounts */
+ FS_MAX_USER_MOUNTS=22, /* int:maximum number of user mounts */
  FS_OCFS2=988, /* ocfs2 */
 };

Index: linux/kernel/sysctl.c
===
--- linux.orig/kernel/sysctl.c 2007-04-11 18:27:46.000000000 +0200
+++ linux/kernel/sysctl.c 2007-04-11 20:08:16.000000000 +0200
@@ -1063,6 +1063,22 @@ static ctl_table fs_table[] = {
 #endif
 #endif
  {
+ .ctl_name = FS_NR_USER_MOUNTS,
+ .procname = "nr_user_mounts",
+ .data = &nr_user_mounts,

```
+  .maxlen  = sizeof(int),
+  .mode  = 0444,
+  .proc_handler = &proc_dointvec,
+ },
+ {
+  .ctl_name = FS_MAX_USER_MOUNTS,
+  .procname = "max_user_mounts",
+  .data  = &max_user_mounts,
+  .maxlen  = sizeof(int),
+  .mode  = 0644,
+  .proc_handler = &proc_dointvec,
+ },
+ {
   .ctl_name = KERN_SETUID_DUMPABLE,
   .procname = "suid_dumpable",
   .data  = &suid_dumpable,
Index: linux/Documentation/filesystems/proc.txt
===================================================================
--- linux.orig/Documentation/filesystems/proc.txt 2007-04-11 18:27:44.000000000 +0200
+++ linux/Documentation/filesystems/proc.txt 2007-04-12 13:32:14.000000000 +0200
@@ -923,6 +923,15 @@ reaches aio-max-nr then io_setup will fa
 raising aio-max-nr does not result in the pre-allocation or re-sizing
 of any kernel data structures.

+nr_user_mounts and max_user_mounts
+----------------------------------
+
+These represent the number of "user" mounts and the maximum number of
+"user" mounts respectively.  User mounts may be created by
+unprivileged users.  User mounts may also be created with sysadmin
+privileges on behalf of a user, in which case nr_user_mounts may
+exceed max_user_mounts.
+
 2.2 /proc/sys/fs/binfmt_misc - Miscellaneous binary formats
 ------------------------------------------------------------

Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-11 20:08:05.000000000 +0200
+++ linux/fs/namespace.c 2007-04-12 13:29:05.000000000 +0200
@@ -39,6 +39,9 @@ static int hash_mask __read_mostly, hash
 static struct kmem_cache *mnt_cache __read_mostly;
 static struct rw_semaphore namespace_sem;

+int nr_user_mounts;
+int max_user_mounts = 1024;
+
 /* /sys/fs */
```

```
 decl_subsys(fs, NULL, NULL);
 EXPORT_SYMBOL_GPL(fs_subsys);
@@ -227,11 +230,30 @@ static struct vfsmount *skip_mnt_tree(st
  return p;
 }

+static void dec_nr_user_mounts(void)
+{
+ spin_lock(&vfsmount_lock);
+ nr_user_mounts--;
+ spin_unlock(&vfsmount_lock);
+}
+
 static void set_mnt_user(struct vfsmount *mnt)
 {
  BUG_ON(mnt->mnt_flags & MNT_USER);
  mnt->mnt_uid = current->uid;
  mnt->mnt_flags |= MNT_USER;
+ spin_lock(&vfsmount_lock);
+ nr_user_mounts++;
+ spin_unlock(&vfsmount_lock);
+}
+
+static void clear_mnt_user(struct vfsmount *mnt)
+{
+ if (mnt->mnt_flags & MNT_USER) {
+  mnt->mnt_uid = 0;
+  mnt->mnt_flags &= ~MNT_USER;
+  dec_nr_user_mounts();
+ }
 }

 static struct vfsmount *clone_mnt(struct vfsmount *old, struct dentry *root,
@@ -283,6 +305,7 @@ static inline void __mntput(struct vfsmo
 {
  struct super_block *sb = mnt->mnt_sb;
  dput(mnt->mnt_root);
+ clear_mnt_user(mnt);
  free_vfsmnt(mnt);
  deactivate_super(sb);
 }
@@ -1023,6 +1046,7 @@ static int do_remount(struct nameidata *
  down_write(&sb->s_umount);
  err = do_remount_sb(sb, flags, data, 0);
  if (!err) {
+  clear_mnt_user(nd->mnt);
   nd->mnt->mnt_flags = mnt_flags;
   if (flags & MS_SETUSER)
```

```
    set_mnt_user(nd->mnt);
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-11 20:07:51.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-12 13:26:42.000000000 +0200
@@ -50,6 +50,9 @@ extern struct inodes_stat_t inodes_stat;

 extern int leases_enable, lease_break_time;

+extern int nr_user_mounts;
+extern int max_user_mounts;
+
 #ifdef CONFIG_DNOTIFY
 extern int dir_notify_enable;
 #endif

--
```

Subject: [patch 04/10] add "permit user mounts" flag to namespaces
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:45 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

If MNT_NS_PERMIT_USERMOUNTS flag is not set for the current namespace,
then unprivileged mounts will be denied.

By default this flag is cleared in all namespaces.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-12 16:50:16.000000000 +0200
+++ linux/fs/namespace.c 2007-04-12 16:50:17.000000000 +0200
@@ -1526,6 +1526,19 @@ dput_out:
  return retval;
 }

+static struct mnt_namespace *alloc_ns(void)
+{
+ struct mnt_namespace *ns;
```

```
+
+ ns = kzalloc(sizeof(struct mnt_namespace), GFP_KERNEL);
+ if (ns) {
+  atomic_set(&ns->count, 1);
+  INIT_LIST_HEAD(&ns->list);
+  init_waitqueue_head(&ns->poll);
+ }
+ return ns;
+}
+
 /*
  * Allocate a new namespace structure and populate it with contents
  * copied from the namespace of the passed in task structure.
@@ -1537,15 +1550,10 @@ static struct mnt_namespace *dup_mnt_ns(
  struct vfsmount *rootmnt = NULL, *pwdmnt = NULL, *altrootmnt = NULL;
  struct vfsmount *p, *q;

- new_ns = kmalloc(sizeof(struct mnt_namespace), GFP_KERNEL);
+ new_ns = alloc_ns();
  if (!new_ns)
   return NULL;

- atomic_set(&new_ns->count, 1);
- INIT_LIST_HEAD(&new_ns->list);
- init_waitqueue_head(&new_ns->poll);
- new_ns->event = 0;
-
  down_write(&namespace_sem);
  /* First pass: copy the tree topology */
  new_ns->root = copy_tree(mnt_ns->root, mnt_ns->root->mnt_root,
@@ -1860,13 +1868,10 @@ static void __init init_mount_tree(void)
  mnt = do_kern_mount("rootfs", 0, "rootfs", NULL);
  if (IS_ERR(mnt))
   panic("Can't create rootfs");
- ns = kmalloc(sizeof(*ns), GFP_KERNEL);
+ ns = alloc_ns();
  if (!ns)
   panic("Can't allocate initial namespace");
- atomic_set(&ns->count, 1);
- INIT_LIST_HEAD(&ns->list);
- init_waitqueue_head(&ns->poll);
- ns->event = 0;
+
  list_add(&mnt->mnt_list, &ns->list);
  ns->root = mnt;
  mnt->mnt_ns = ns;
Index: linux/include/linux/mnt_namespace.h
===================================================================
```

```
--- linux.orig/include/linux/mnt_namespace.h 2007-04-12 16:50:02.000000000 +0200
+++ linux/include/linux/mnt_namespace.h 2007-04-12 16:50:17.000000000 +0200
@@ -6,12 +6,16 @@
 #include <linux/sched.h>
 #include <linux/nsproxy.h>

+/* mnt_namespace flags */
+#define MNT_NS_PERMIT_USERMOUNTS (1 << 0)
+
 struct mnt_namespace {
  atomic_t  count;
  struct vfsmount * root;
  struct list_head list;
  wait_queue_head_t poll;
  int event;
+ int flags;
 };

 extern struct mnt_namespace *copy_mnt_ns(int, struct mnt_namespace *,

--
```

Subject: [patch 05/10] add "permit user mounts in new namespace" clone flag
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:46 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

If CLONE_NEWNS and CLONE_NEWNS_USERMNT are given to clone(2) or
unshare(2), then allow user mounts within the new namespace.

This is not flexible enough, because user mounts can't be enabled for
the initial namespace.

The remaining clone bits also getting dangerously few...

Alternatives are:

 - prctl() flag
 - setting through the containers filesystem

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-12 13:46:19.000000000 +0200
+++ linux/fs/namespace.c 2007-04-12 13:54:36.000000000 +0200
@@ -1617,6 +1617,8 @@ struct mnt_namespace *copy_mnt_ns(int fl
   return ns;

  new_ns = dup_mnt_ns(ns, new_fs);
+ if (new_ns && (flags & CLONE_NEWNS_USERMNT))
+  new_ns->flags |= MNT_NS_PERMIT_USERMOUNTS;

  put_mnt_ns(ns);
  return new_ns;
Index: linux/include/linux/sched.h
===================================================================
--- linux.orig/include/linux/sched.h 2007-04-12 13:26:48.000000000 +0200
+++ linux/include/linux/sched.h 2007-04-12 13:54:36.000000000 +0200
@@ -26,6 +26,7 @@
 #define CLONE_STOPPED  0x02000000 /* Start in stopped state */
 #define CLONE_NEWUTS  0x04000000 /* New utsname group? */
 #define CLONE_NEWIPC  0x08000000 /* New ipcs */
+#define CLONE_NEWNS_USERMNT 0x10000000 /* Allow user mounts in ns? */

 /*
  * Scheduling policies
Index: linux/kernel/fork.c
===================================================================
--- linux.orig/kernel/fork.c 2007-04-11 18:27:46.000000000 +0200
+++ linux/kernel/fork.c 2007-04-12 13:59:10.000000000 +0200
@@ -1586,7 +1586,7 @@ asmlinkage long sys_unshare(unsigned lon
  err = -EINVAL;
  if (unshare_flags & ~(CLONE_THREAD|CLONE_FS|CLONE_NEWNS|CLONE_SIGHAND|
     CLONE_VM|CLONE_FILES|CLONE_SYSVSEM|
-    CLONE_NEWUTS|CLONE_NEWIPC))
+    CLONE_NEWUTS|CLONE_NEWIPC|CLONE_NEWNS_USERMNT))
   goto bad_unshare_out;

  if ((err = unshare_thread(unshare_flags)))

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

Subject: [patch 06/10] propagate error values from clone_mnt
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:47 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Allow clone_mnt() to return errors other than ENOMEM. This will be
used for returning a different error value when the number of user
mounts goes over the limit.

Fix copy_tree() to return EPERM for unbindable mounts.

Don't propagate further from dup_mnt_ns() as that copy_tree() can only
fail with -ENOMEM.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/namespace.c
===
--- linux.orig/fs/namespace.c 2007-04-12 13:54:36.000000000 +0200
+++ linux/fs/namespace.c 2007-04-12 14:04:25.000000000 +0200
@@ -261,42 +261,42 @@ static struct vfsmount *clone_mnt(struct
 {
  struct super_block *sb = old->mnt_sb;
  struct vfsmount *mnt = alloc_vfsmnt(old->mnt_devname);
+ if (!mnt)
+ return ERR_PTR(-ENOMEM);

- if (mnt) {
- mnt->mnt_flags = old->mnt_flags;
- atomic_inc(&sb->s_active);
- mnt->mnt_sb = sb;
- mnt->mnt_root = dget(root);
- 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;
-   CLEAR_MNT_SHARED(mnt);
- } else {
-   if ((flag & CL_PROPAGATION) || IS_MNT_SHARED(old))
-    list_add(&mnt->mnt_share, &old->mnt_share);

```
-   if (IS_MNT_SLAVE(old))
-    list_add(&mnt->mnt_slave, &old->mnt_slave);
-   mnt->mnt_master = old->mnt_master;
-  }
-  if (flag & CL_MAKE_SHARED)
-   set_mnt_shared(mnt);
+ mnt->mnt_flags = old->mnt_flags;
+ atomic_inc(&sb->s_active);
+ mnt->mnt_sb = sb;
+ mnt->mnt_root = dget(root);
+ 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);

-  /* stick the duplicate mount on the same expiry list
-   * as the original if that was on one */
-  if (flag & CL_EXPIRE) {
-   spin_lock(&vfsmount_lock);
-   if (!list_empty(&old->mnt_expire))
-    list_add(&mnt->mnt_expire, &old->mnt_expire);
-   spin_unlock(&vfsmount_lock);
-  }
+ if (flag & CL_SLAVE) {
+  list_add(&mnt->mnt_slave, &old->mnt_slave_list);
+  mnt->mnt_master = old;
+  CLEAR_MNT_SHARED(mnt);
+ } else {
+  if ((flag & CL_PROPAGATION) || IS_MNT_SHARED(old))
+   list_add(&mnt->mnt_share, &old->mnt_share);
+  if (IS_MNT_SLAVE(old))
+   list_add(&mnt->mnt_slave, &old->mnt_slave);
+  mnt->mnt_master = old->mnt_master;
+ }
+ if (flag & CL_MAKE_SHARED)
+  set_mnt_shared(mnt);
+
+ /* stick the duplicate mount on the same expiry list
+  * as the original if that was on one */
+ if (flag & CL_EXPIRE) {
+  spin_lock(&vfsmount_lock);
+  if (!list_empty(&old->mnt_expire))
+   list_add(&mnt->mnt_expire, &old->mnt_expire);
+  spin_unlock(&vfsmount_lock);
  }
```

```
  return mnt;
 }
@@ -781,11 +781,11 @@ struct vfsmount *copy_tree(struct vfsmou
  struct nameidata nd;

  if (!(flag & CL_COPY_ALL) && IS_MNT_UNBINDABLE(mnt))
-  return NULL;
+  return ERR_PTR(-EPERM);

  res = q = clone_mnt(mnt, dentry, flag);
- if (!q)
-  goto Enomem;
+ if (IS_ERR(q))
+  goto error;
  q->mnt_mountpoint = mnt->mnt_mountpoint;

  p = mnt;
@@ -806,8 +806,8 @@ struct vfsmount *copy_tree(struct vfsmou
    nd.mnt = q;
    nd.dentry = p->mnt_mountpoint;
    q = clone_mnt(p, p->mnt_root, flag);
-   if (!q)
-    goto Enomem;
+   if (IS_ERR(q))
+    goto error;
    spin_lock(&vfsmount_lock);
    list_add_tail(&q->mnt_list, &res->mnt_list);
    attach_mnt(q, &nd);
@@ -815,7 +815,7 @@ struct vfsmount *copy_tree(struct vfsmou
   }
  }
  return res;
-Enomem:
+ error:
  if (res) {
   LIST_HEAD(umount_list);
   spin_lock(&vfsmount_lock);
@@ -823,7 +823,7 @@ Enomem:
   spin_unlock(&vfsmount_lock);
   release_mounts(&umount_list);
  }
- return NULL;
+ return q;
 }

 /*
@@ -999,13 +999,13 @@ static int do_loopback(struct nameidata
   goto out;
```

```
  clone_flags = (flags & MS_SETUSER) ? CL_SETUSER : 0;
- err = -ENOMEM;
  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, clone_flags);

- if (!mnt)
+ err = PTR_ERR(mnt);
+ if (IS_ERR(mnt))
   goto out;

  err = graft_tree(mnt, nd);
@@ -1558,7 +1558,7 @@ static struct mnt_namespace *dup_mnt_ns(
  /* First pass: copy the tree topology */
  new_ns->root = copy_tree(mnt_ns->root, mnt_ns->root->mnt_root,
      CL_COPY_ALL | CL_EXPIRE);
- if (!new_ns->root) {
+ if (IS_ERR(new_ns->root)) {
   up_write(&namespace_sem);
   kfree(new_ns);
   return NULL;
Index: linux/fs/pnode.c
===================================================================
--- linux.orig/fs/pnode.c 2007-04-12 13:26:42.000000000 +0200
+++ linux/fs/pnode.c 2007-04-12 14:04:25.000000000 +0200
@@ -187,8 +187,9 @@ int propagate_mnt(struct vfsmount *dest_

   source =  get_source(m, prev_dest_mnt, prev_src_mnt, &type);

-  if (!(child = copy_tree(source, source->mnt_root, type))) {
-   ret = -ENOMEM;
+  child = copy_tree(source, source->mnt_root, type);
+  if (IS_ERR(child)) {
+   ret = PTR_ERR(child);
    list_splice(tree_list, tmp_list.prev);
    goto out;
   }

--
```

Subject: [patch 07/10] allow unprivileged bind mounts
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:48 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Allow bind mounts to unprivileged users if the following conditions
are met:

 - user mounts are permitted in the current mount namespace
 - mountpoint is not a symlink or special file
 - mountpoint is not a sticky directory or is owned by the current user
 - mountpoint is writable by user
 - the number of user mounts is below the maximum

Unprivileged mounts imply MS_SETUSER, and will also have the "nosuid"
and "nodev" mount flags set.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/namespace.c
===
--- linux.orig/fs/namespace.c 2007-04-12 14:04:25.000000000 +0200
+++ linux/fs/namespace.c 2007-04-12 14:04:27.000000000 +0200
@@ -237,11 +237,30 @@ static void dec_nr_user_mounts(void)
  spin_unlock(&vfsmount_lock);
 }

-static void set_mnt_user(struct vfsmount *mnt)
+static int reserve_user_mount(void)
+{
+ int err = 0;
+ spin_lock(&vfsmount_lock);
+ if (nr_user_mounts >= max_user_mounts && !capable(CAP_SYS_ADMIN))
+ err = -EPERM;
+ else
+ nr_user_mounts++;
+ spin_unlock(&vfsmount_lock);
+ return err;
+}
+
+static void __set_mnt_user(struct vfsmount *mnt)
 {
  BUG_ON(mnt->mnt_flags & MNT_USER);
  mnt->mnt_uid = current->uid;
  mnt->mnt_flags |= MNT_USER;
+ if (!capable(CAP_SYS_ADMIN))
+ mnt->mnt_flags |= MNT_NOSUID | MNT_NODEV;

```
+}
+
+static void set_mnt_user(struct vfsmount *mnt)
+{
+ __set_mnt_user(mnt);
  spin_lock(&vfsmount_lock);
  nr_user_mounts++;
  spin_unlock(&vfsmount_lock);
@@ -260,9 +279,16 @@ static struct vfsmount *clone_mnt(struct
       int flag)
 {
  struct super_block *sb = old->mnt_sb;
- struct vfsmount *mnt = alloc_vfsmnt(old->mnt_devname);
+ struct vfsmount *mnt;
+
+ if (flag & CL_SETUSER) {
+  int err = reserve_user_mount();
+  if (err)
+   return ERR_PTR(err);
+ }
+ mnt = alloc_vfsmnt(old->mnt_devname);
  if (!mnt)
-  return ERR_PTR(-ENOMEM);
+  goto alloc_failed;

  mnt->mnt_flags = old->mnt_flags;
  atomic_inc(&sb->s_active);
@@ -274,7 +300,7 @@ static struct vfsmount *clone_mnt(struct
  /* don't copy the MNT_USER flag */
  mnt->mnt_flags &= ~MNT_USER;
  if (flag & CL_SETUSER)
-  set_mnt_user(mnt);
+  __set_mnt_user(mnt);

  if (flag & CL_SLAVE) {
   list_add(&mnt->mnt_slave, &old->mnt_slave_list);
@@ -299,6 +325,11 @@ static struct vfsmount *clone_mnt(struct
   spin_unlock(&vfsmount_lock);
  }
  return mnt;
+
+ alloc_failed:
+ if (flag & CL_SETUSER)
+  dec_nr_user_mounts();
+ return ERR_PTR(-ENOMEM);
 }

 static inline void __mntput(struct vfsmount *mnt)
```

```
@@ -745,22 +776,35 @@ asmlinkage long sys_oldumount(char __use

 #endif

-static int mount_is_safe(struct nameidata *nd)
+/*
+ * Conditions for unprivileged mounts are:
+ * - user mounts are permitted in the current mount namespace
+ * - mountpoint is not a symlink or special file
+ * - mountpoint is "absolutely" writable by user
+ *  o if it's a sticky directory, it must be owned by the user
+ *  o it must not be an append-only file/directory
+ */
+static int mount_is_safe(struct nameidata *nd, int *flags)
 {
+ struct inode *inode = nd->dentry->d_inode;
+
  if (capable(CAP_SYS_ADMIN))
   return 0;
- return -EPERM;
-#ifdef notyet
- if (S_ISLNK(nd->dentry->d_inode->i_mode))
+
+ if (!(current->nsproxy->mnt_ns->flags & MNT_NS_PERMIT_USERMOUNTS))
   return -EPERM;
- if (nd->dentry->d_inode->i_mode & S_ISVTX) {
-  if (current->uid != nd->dentry->d_inode->i_uid)
-   return -EPERM;
- }
- if (vfs_permission(nd, MAY_WRITE))
+
+ if (!S_ISDIR(inode->i_mode) && !S_ISREG(inode->i_mode))
+  return -EPERM;
+
+ if ((inode->i_mode & S_ISVTX) && current->fsuid != inode->i_uid)
   return -EPERM;
+
+ if (vfs_permission(nd, MAY_WRITE) || IS_APPEND(inode))
+  return -EPERM;
+
+ *flags |= MS_SETUSER;
  return 0;
-#endif
 }

 static int lives_below_in_same_fs(struct dentry *d, struct dentry *dentry)
@@ -981,7 +1025,7 @@ static int do_loopback(struct nameidata
  int clone_flags;
```

```
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
- int err = mount_is_safe(nd);
+ int err = mount_is_safe(nd, &flags);
  if (err)
   return err;
  if (!old_name || !*old_name)
```

--

_____

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

---

Subject: [patch 08/10] put declaration of put_filesystem() in fs.h
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:49 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Declarations go into headers.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/super.c
```
===================================================================
--- linux.orig/fs/super.c 2007-04-12 13:26:42.000000000 +0200
+++ linux/fs/super.c 2007-04-12 14:04:29.000000000 +0200
@@ -40,10 +40,6 @@
 #include <asm/uaccess.h>


-void get_filesystem(struct file_system_type *fs);
-void put_filesystem(struct file_system_type *fs);
-struct file_system_type *get_fs_type(const char *name);
-
 LIST_HEAD(super_blocks);
 DEFINE_SPINLOCK(sb_lock);
```

Index: linux/include/linux/fs.h
```
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-12 13:26:42.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-12 14:04:29.000000000 +0200
@@ -1918,6 +1918,8 @@ extern int vfs_fstat(unsigned int, struc

 extern int vfs_ioctl(struct file *, unsigned int, unsigned int, unsigned long);
```

```
+extern void get_filesystem(struct file_system_type *fs);
+extern void put_filesystem(struct file_system_type *fs);
 extern struct file_system_type *get_fs_type(const char *name);
 extern struct super_block *get_super(struct block_device *);
 extern struct super_block *user_get_super(dev_t);

--
```

_____

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

---

Subject: [patch 09/10] allow unprivileged mounts
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:50 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Define a new fs flag FS_SAFE, which denotes, that unprivileged
mounting of this filesystem may not constitute a security problem.

Since most filesystems haven't been designed with unprivileged
mounting in mind, a thorough audit is needed before setting this flag.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

```
Index: linux/fs/namespace.c
===================================================================
--- linux.orig/fs/namespace.c 2007-04-12 14:04:27.000000000 +0200
+++ linux/fs/namespace.c 2007-04-12 14:04:32.000000000 +0200
@@ -784,7 +784,8 @@ asmlinkage long sys_oldumount(char __use
  *  o if it's a sticky directory, it must be owned by the user
  *  o it must not be an append-only file/directory
  */
-static int mount_is_safe(struct nameidata *nd, int *flags)
+static int mount_is_safe(struct nameidata *nd, struct file_system_type *type,
+    int *flags)
 {
  struct inode *inode = nd->dentry->d_inode;

@@ -794,6 +795,9 @@ static int mount_is_safe(struct nameidat
  if (!(current->nsproxy->mnt_ns->flags & MNT_NS_PERMIT_USERMOUNTS))
   return -EPERM;

+ if (type && !(type->fs_flags & FS_SAFE))
```

```
+  return -EPERM;
+
  if (!S_ISDIR(inode->i_mode) && !S_ISREG(inode->i_mode))
   return -EPERM;

@@ -1025,7 +1029,7 @@ static int do_loopback(struct nameidata
  int clone_flags;
  struct nameidata old_nd;
  struct vfsmount *mnt = NULL;
- int err = mount_is_safe(nd, &flags);
+ int err = mount_is_safe(nd, NULL, &flags);
  if (err)
   return err;
  if (!old_name || !*old_name)
@@ -1187,26 +1191,46 @@ out:
  * create a new mount for userspace and request it to be added into the
  * namespace's tree
  */
-static int do_new_mount(struct nameidata *nd, char *type, int flags,
+static int do_new_mount(struct nameidata *nd, char *fstype, int flags,
    int mnt_flags, char *name, void *data)
 {
+ int err;
  struct vfsmount *mnt;
+ struct file_system_type *type;

- if (!type || !memchr(type, 0, PAGE_SIZE))
+ if (!fstype || !memchr(fstype, 0, PAGE_SIZE))
   return -EINVAL;

- /* we need capabilities... */
- if (!capable(CAP_SYS_ADMIN))
-  return -EPERM;
+ type = get_fs_type(fstype);
+ if (!type)
+  return -ENODEV;

- mnt = do_kern_mount(type, flags & ~MS_SETUSER, name, data);
- if (IS_ERR(mnt))
+ err = mount_is_safe(nd, type, &flags);
+ if (err)
+  goto out_put_filesystem;
+
+ if (flags & MS_SETUSER) {
+  err = reserve_user_mount();
+  if (err)
+   goto out_put_filesystem;
+ }
```

```
+
+ mnt = vfs_kern_mount(type, flags & ~MS_SETUSER, name, data);
+ put_filesystem(type);
+ if (IS_ERR(mnt)) {
+  if (flags & MS_SETUSER)
+   dec_nr_user_mounts();
   return PTR_ERR(mnt);
+ }

  if (flags & MS_SETUSER)
-  set_mnt_user(mnt);
+  __set_mnt_user(mnt);

  return do_add_mount(mnt, nd, mnt_flags, NULL);
+
+ out_put_filesystem:
+ put_filesystem(type);
+ return err;
 }

 /*
@@ -1236,7 +1260,7 @@ int do_add_mount(struct vfsmount *newmnt
  if (S_ISLNK(newmnt->mnt_root->d_inode->i_mode))
   goto unlock;

- /* MNT_USER was set earlier */
+ /* some flags may have been set earlier */
  newmnt->mnt_flags |= mnt_flags;
  if ((err = graft_tree(newmnt, nd)))
   goto unlock;
Index: linux/include/linux/fs.h
===================================================================
--- linux.orig/include/linux/fs.h 2007-04-12 14:04:29.000000000 +0200
+++ linux/include/linux/fs.h 2007-04-12 14:04:32.000000000 +0200
@@ -96,6 +96,7 @@ extern int dir_notify_enable;
 #define FS_REQUIRES_DEV 1
 #define FS_BINARY_MOUNTDATA 2
 #define FS_HAS_SUBTYPE 4
+#define FS_SAFE 8  /* Safe to mount by unprivileged users */
 #define FS_REVAL_DOT 16384 /* Check the paths ".", ".." for staleness */
 #define FS_RENAME_DOES_D_MOVE 32768 /* FS will handle d_move()
       * during rename() internally.

--
```

Subject: [patch 10/10] allow unprivileged fuse mounts
Posted by Miklos Szeredi on Thu, 12 Apr 2007 16:45:51 GMT
View Forum Message <> Reply to Message

From: Miklos Szeredi <mszeredi@suse.cz>

Use FS_SAFE for "fuse" fs type, but not for "fuseblk".

FUSE was designed from the beginning to be safe for unprivileged users. This has also been verified in practice over many years. And unprivileged fuse mounts still require a private namespace with user mounts enabled, this is more strict than the current userspace policy.

This will enable future installations to remove the suid-root fusermount utility.

Don't require the "user_id=" and "group_id=" options for unprivileged mounts, but if they are present verify them for sanity.

Disallow the "allow_other" option for unprivileged mounts.

Signed-off-by: Miklos Szeredi <mszeredi@suse.cz>

Index: linux/fs/fuse/inode.c
```
===================================================================
--- linux.orig/fs/fuse/inode.c 2007-04-12 13:26:42.000000000 +0200
+++ linux/fs/fuse/inode.c 2007-04-12 14:04:34.000000000 +0200
@@ -311,6 +311,19 @@ static int parse_fuse_opt(char *opt, str
  d->max_read = ~0;
  d->blksize = 512;

+ /*
+  * For unprivileged mounts use current uid/gid.  Still allow
+  * "user_id" and "group_id" options for compatibility, but
+  * only if they match these values.
+  */
+ if (!capable(CAP_SYS_ADMIN)) {
+  d->user_id = current->uid;
+  d->user_id_present = 1;
+  d->group_id = current->gid;
+  d->group_id_present = 1;
+
+ }
+
  while ((p = strsep(&opt, ",")) != NULL) {
   int token;
   int value;
@@ -339,6 +352,8 @@ static int parse_fuse_opt(char *opt, str
```

```
   case OPT_USER_ID:
    if (match_int(&args[0], &value))
     return 0;
+    if (d->user_id_present && d->user_id != value)
+     return 0;
    d->user_id = value;
    d->user_id_present = 1;
    break;
@@ -346,6 +361,8 @@ static int parse_fuse_opt(char *opt, str
   case OPT_GROUP_ID:
    if (match_int(&args[0], &value))
     return 0;
+    if (d->group_id_present && d->group_id != value)
+     return 0;
    d->group_id = value;
    d->group_id_present = 1;
    break;
@@ -536,6 +553,10 @@ static int fuse_fill_super(struct super_
  if (!parse_fuse_opt((char *) data, &d, is_bdev))
   return -EINVAL;

+ /* This is a privileged option */
+ if ((d.flags & FUSE_ALLOW_OTHER) && !capable(CAP_SYS_ADMIN))
+  return -EPERM;
+
  if (is_bdev) {
 #ifdef CONFIG_BLOCK
   if (!sb_set_blocksize(sb, d.blksize))
@@ -639,6 +660,7 @@ static struct file_system_type fuse_fs_t
  .fs_flags = FS_HAS_SUBTYPE,
  .get_sb  = fuse_get_sb,
  .kill_sb = kill_anon_super,
+ .fs_flags = FS_SAFE,
 };

 #ifdef CONFIG_BLOCK

--
```


Subject: Re: [patch 05/10] add "permit user mounts in new namespace" clone flag
Posted by serue on Fri, 13 Apr 2007 13:47:33 GMT
View Forum Message <> Reply to Message

Quoting Miklos Szeredi (miklos@szeredi.hu):
> > Given the existence of shared subtrees allowing/denying this at the mount
> > namespace level is silly and wrong.
> >
> > If we need more than just the filesystem permission checks can we
> > make it a mount flag settable with mount and remount that allows
> > non-privileged users the ability to create mount points under it
> > in directories they have full read/write access to.
>
> OK, that makes sense.
>
> > I don't like the use of clone flags for this purpose but in this
> > case the shared subtress are a much more fundamental reasons for not
> > doing this at the namespace level.
>
> I'll drop the clone flag, and add a mount flag instead.
>
> Thanks,
> Miklos

Makes sense, so then on login pam has to spawn a new user namespace and
construct a root fs with no shared subtrees and with the
user-mounts-allowed flag specified?

-serge

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

---

Subject: Re: [patch 05/10] add "permit user mounts in new namespace" clone flag
Posted by ebiederm on Fri, 13 Apr 2007 14:22:10 GMT
View Forum Message <> Reply to Message

"Serge E. Hallyn" <serue@us.ibm.com> writes:

> Quoting Miklos Szeredi (miklos@szeredi.hu):
>> > Given the existence of shared subtrees allowing/denying this at the mount
>> > namespace level is silly and wrong.
>> >
>> > If we need more than just the filesystem permission checks can we
>> > make it a mount flag settable with mount and remount that allows
>> > non-privileged users the ability to create mount points under it
>> > in directories they have full read/write access to.
>>
>> OK, that makes sense.
>>

>> > I don't like the use of clone flags for this purpose but in this
>> > case the shared subtress are a much more fundamental reasons for not
>> > doing this at the namespace level.
>>
>> I'll drop the clone flag, and add a mount flag instead.
>>
>> Thanks,
>> Miklos
>
> Makes sense, so then on login pam has to spawn a new user namespace and
> construct a root fs with no shared subtrees and with the
> user-mounts-allowed flag specified?

I was expecting the usage in the normal case to be the Al Viro style
with shared subtrees setup for each user, with the shared subtree marked
with user-mounts-allowed. Then on login pam would unshare the namespace
and restrict the user to their specific portion of the shared subtree.

If you don't use multiple mount namespaces all of the users have to
agree on what they want the non-privileged part of the namespace
to look like.

If you don't use shared subtrees you have to deal with all of the
joys of implementing enter, or else multiple logins from the same user
have problems.

Eric

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

---

Subject: Re: [patch 02/10] allow unprivileged umount
Posted by Greg KH on Mon, 16 Apr 2007 19:58:10 GMT
View Forum Message <> Reply to Message

On Mon, Apr 16, 2007 at 01:39:19PM -0600, Eric W. Biederman wrote:
> Miklos Szeredi <miklos@szeredi.hu> writes:
>
> > From: Miklos Szeredi <mszeredi@suse.cz>
> >
> > The owner doesn't need sysadmin capabilities to call umount().
> >
> > Similar behavior as umount(8) on mounts having "user=UID" option in
> > /etc/mtab. The difference is that umount also checks /etc/fstab,
> > presumably to exclude another mount on the same mountpoint.
> >

>
> bool in the kernel?

It's there already...

> int would be much more recognizable as this is not C++
>
> Or do you have place to convert the rest of the kernel that is using
> int to return a true/false value to bool?

It's already underway:
 $ git log | grep bool | wc -l
 123