
Subject: Re: How to query mount propagation state?
Posted by [Ram Pai](#) on Mon, 16 Apr 2007 17:39:46 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Mon, 2007-04-16 at 12:34 +0200, Miklos Szeredi wrote:
> Currently one of the difficulties with mount propagations is that
> there's no way to know the current state of the propagation tree.
>
> Has anyone thought about how this info could be queried from
> userspace?

I am attaching two patches that I had done way back in Oct 2006 with Al Viro. I had sent these patches to Al Viro. But I forgot to follow them up, I guess so did Al Viro.

The first patch disambiguates multiple mount-instances of the same filesystem (or part of the same filesystem), by introducing a new interface `/proc/mounts_new`.

The second patch introduces a new proc interface that exposes all the propagation trees within a namespace. It does not show propagated mounts residing in a different namespace (for privacy reasons). Maybe one could modify the patch a little, to allow it; if the user has root privileges.

RP

PS: Sorry these are attachments instead of inline patches. I am scared of inlining in evolution. If needed I can send inline patches through mutt.

>
> Thanks,
> Miklos

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

File Attachments

- 1) [mounts.patch](#), downloaded 302 times
 - 2) [propagation.patch](#), downloaded 333 times
-

Subject: Re: How to query mount propagation state?
Posted by [serue](#) on Mon, 16 Apr 2007 19:16:00 GMT

Quoting Ram Pai (linuxram@us.ibm.com):

> On Mon, 2007-04-16 at 12:34 +0200, Miklos Szeredi wrote:

> > Currently one of the difficulties with mount propagations is that
> > there's no way to know the current state of the propagation tree.

> >

> > Has anyone thought about how this info could be queried from
> > userspace?

>

> I am attaching two patches that I had done way back in Oct 2006
> with Al Viro. I had sent these patches to Al Viro. But I forgot to
> follow them up, I guess so did Al Viro.

>

> The first patch disambiguates multiple mount-instances of the same
> filesystem (or part of the same filesystem), by introducing a new
> interface /proc/mounts_new.

>

> The second patch introduces a new proc interface that exposes all the
> propagation trees within a namespace. It does not show propagated
> mounts residing in a different namespace (for privacy reasons). Maybe
> one could modify the patch a little, to allow it; if the user has
> root privileges.

>

> RP

>

> PS: Sorry these are attachments instead of inline patches. I am scared
> of inlining in evolution. If needed I can send inline patches through
> mutt.

>

> >

> > Thanks,

> > Miklos

> This patch disambiguates multiple mount-instances of the same
> filesystem (or part of the same filesystem), by introducing a new
> interface /proc/mounts_new. The interface has the following format.

>

> -----

> FSID mntpt root-dentry fstype fs-options

> -----

>

> NOTE: root-dentry is the path to the dentry w.r.t to the root dentry of the
> same filesystem.

>

> for example: lets say we attempt the following commands

> mount --bind /var /mnt

> mount --bind /mnt/tmp /tmp1

>

```

> 'cat /proc/mounts' shows the following:
> /dev/root /mnt ext2 rw 0 0
> /dev/root /tmp1 ext2 rw 0 0
>
> NOTE: The above mount entries, do not indicate that /tmp1 contains the same
> directory tree as /var/tmp.
>
> But 'cat /proc/mounts_new' shows us the following:
> 0x6200 /mnt /var ext2 rw 0 0
> 0x6200 /tmp1 /var/tmp ext2 rw 0 0
>
> The above entries clearly indicates that /var/tmp directory of the ext2
> filesystem with fsid=0x6200 is the directory tree that resides under /tmp1
>
> Signed-off-by: Ram Pai <linuxram@us.ibm.com>
>
> ---
> fs/dcache.c          | 53 ++++++
> fs/namespace.c      | 35 ++++++
> fs/proc/base.c      | 32 ++++++
> fs/proc/proc_misc.c | 1
> fs/seq_file.c       | 77 ++++++
> include/linux/dcache.h | 1
> include/linux/seq_file.h | 1
> 7 files changed, 172 insertions(+), 28 deletions(-)
>
> Index: linux-2.6.17.10/fs/proc/base.c
> =====
> --- linux-2.6.17.10.orig/fs/proc/base.c
> +++ linux-2.6.17.10/fs/proc/base.c
> @@ -104,6 +104,7 @@ enum pid_directory_inos {
>  PROC_TGID_MAPS,
>  PROC_TGID_NUMA_MAPS,
>  PROC_TGID_MOUNTS,
> + PROC_TGID_MOUNTS_NEW,
>  PROC_TGID_MOUNTSTATS,
>  PROC_TGID_WCHAN,
> #ifdef CONFIG_MMU
> @@ -145,6 +146,7 @@ enum pid_directory_inos {
>  PROC_TID_MAPS,
>  PROC_TID_NUMA_MAPS,
>  PROC_TID_MOUNTS,
> + PROC_TID_MOUNTS_NEW,
>  PROC_TID_MOUNTSTATS,
>  PROC_TID_WCHAN,
> #ifdef CONFIG_MMU
> @@ -203,6 +205,7 @@ static struct pid_entry tgid_base_stuff[
>  E(PROC_TGID_ROOT, "root", S_IFLNK|S_IRWXUGO),

```

```

> E(PROC_TGID_EXE, "exe", S_IFLNK|S_IRWXUGO),
> E(PROC_TGID_MOUNTS, "mounts", S_IFREG|S_IRUGO),
> + E(PROC_TGID_MOUNTS_NEW,"mounts_new", S_IFREG|S_IRUGO),
> E(PROC_TGID_MOUNTSTATS, "mountstats", S_IFREG|S_IRUSR),
> #ifdef CONFIG_MMU
> E(PROC_TGID_SMAPS, "smaps", S_IFREG|S_IRUGO),
> @@ -246,6 +249,7 @@ static struct pid_entry tid_base_stuff[]
> E(PROC_TID_ROOT, "root", S_IFLNK|S_IRWXUGO),
> E(PROC_TID_EXE, "exe", S_IFLNK|S_IRWXUGO),
> E(PROC_TID_MOUNTS, "mounts", S_IFREG|S_IRUGO),
> + E(PROC_TID_MOUNTS_NEW, "mounts_new", S_IFREG|S_IRUGO),
> #ifdef CONFIG_MMU
> E(PROC_TID_SMAPS, "smaps", S_IFREG|S_IRUGO),
> #endif
> @@ -692,13 +696,13 @@ static struct file_operations proc_smaps
> };
> #endif
>
> -extern struct seq_operations mounts_op;
> struct proc_mounts {
> struct seq_file m;
> int event;
> };
>
> -static int mounts_open(struct inode *inode, struct file *file)
> +static int __mounts_open(struct inode *inode, struct file *file,
> + struct seq_operations *mounts_op)
> {
> struct task_struct *task = proc_task(inode);
> struct namespace *namespace;
> @@ -716,7 +720,7 @@ static int mounts_open(struct inode *ino
> p = kmalloc(sizeof(struct proc_mounts), GFP_KERNEL);
> if (p) {
> file->private_data = &p->m;
> - ret = seq_open(file, &mounts_op);
> + ret = seq_open(file, mounts_op);
> if (!ret) {
> p->m.private = namespace;
> p->event = namespace->event;
> @@ -729,6 +733,16 @@ static int mounts_open(struct inode *ino
> return ret;
> }
>
> +extern struct seq_operations mounts_op, mounts_new_op;
> +static int mounts_open(struct inode *inode, struct file *file)
> +{
> + return (__mounts_open(inode, file, &mounts_op));
> +}

```

```

> +static int mounts_new_open(struct inode *inode, struct file *file)
> +{
> + return __mounts_open(inode, file, &mounts_new_op);
> +}
> +
> static int mounts_release(struct inode *inode, struct file *file)
> {
> struct seq_file *m = file->private_data;
> @@ -763,6 +777,14 @@ static struct file_operations proc_mount
> .poll = mounts_poll,
> };
>
> +static struct file_operations proc_mounts_new_operations = {
> + .open = mounts_new_open,
> + .read = seq_read,
> + .llseek = seq_lseek,
> + .release = mounts_release,
> + .poll = mounts_poll,
> +};
> +
> extern struct seq_operations mountstats_op;
> static int mountstats_open(struct inode *inode, struct file *file)
> {
> @@ -1799,6 +1821,10 @@ static struct dentry *proc_pident_lookup
> case PROC_TGID_MOUNTS:
> inode->i_fop = &proc_mounts_operations;
> break;
> + case PROC_TID_MOUNTS_NEW:
> + case PROC_TGID_MOUNTS_NEW:
> + inode->i_fop = &proc_mounts_new_operations;
> + break;
> #ifdef CONFIG_MMU
> case PROC_TID_SMAPS:
> case PROC_TGID_SMAPS:
> Index: linux-2.6.17.10/fs/dcache.c
> =====
> --- linux-2.6.17.10.orig/fs/dcache.c
> +++ linux-2.6.17.10/fs/dcache.c
> @@ -1477,6 +1477,59 @@ char * d_path(struct dentry *dentry, str
> return res;
> }
>
> +static inline int prepend(char **buffer, int *buflen, const char *str,
> + int namelen)
> +{
> + if ((*buflen -= namelen) < 0)
> + return 1;
> + *buffer -= namelen;

```

```

> + memcpy(*buffer, str, namelen);
> + return 0;
> +}
> +
> +/*
> + * write full pathname into buffer and return start of pathname.
> + * If @vfsmnt is not specified return the path relative to the
> + * its filesystem's root.
> + */
> +char * dentry_path(struct dentry *dentry, char *buf, int buflen)
> +{
> + char * end = buf+buflen;
> + char * retval;
> +
> + spin_lock(&dcache_lock);
> + prepend(&end, &buflen, "\0", 1);
> + if (!IS_ROOT(dentry) && d_unhashed(dentry)) {
> + if (prepend(&end, &buflen, "//deleted", 10))
> + goto Elong;
> + }
> + /* Get '/' right */
> + retval = end-1;
> + *retval = '/';
> +
> + for (;;) {
> + struct dentry * parent;
> + if (IS_ROOT(dentry))
> + break;
> +
> + parent = dentry->d_parent;
> + prefetch(parent);
> +
> + if (prepend(&end, &buflen, dentry->d_name.name,
> + dentry->d_name.len) ||
> + prepend(&end, &buflen, "/", 1))
> + goto Elong;
> +
> + retval = end;
> + dentry = parent;
> + }
> + spin_unlock(&dcache_lock);
> + return retval;
> +Elong:
> + spin_unlock(&dcache_lock);
> + return ERR_PTR(-ENAMETOOLONG);
> +}
> +
> /*

```

```

> * NOTE! The user-level library version returns a
> * character pointer. The kernel system call just
> Index: linux-2.6.17.10/fs/seq_file.c
> =====
> --- linux-2.6.17.10.orig/fs/seq_file.c
> +++ linux-2.6.17.10/fs/seq_file.c
> @@ -338,38 +338,75 @@ int seq_printf(struct seq_file *m, const
> }
> EXPORT_SYMBOL(seq_printf);
>
> -int seq_path(struct seq_file *m,
> - struct vfsmount *mnt, struct dentry *dentry,
> +static inline char *mangle_path(char *s, char *p, char *esc)
> +{
> + while (s <= p) {
> + char c = *p++;
> + if (!c) {
> + return s;
> + } else if (!strchr(esc, c)) {
> + *s++ = c;
> + } else if (s + 4 > p) {
> + break;
> + } else {
> + *s++ = '\\';
> + *s++ = '0' + ((c & 0300) >> 6);
> + *s++ = '0' + ((c & 070) >> 3);
> + *s++ = '0' + (c & 07);
> + }
> + }
> + return NULL;
> + }
> + /*
> + * return the absolute path of 'dentry' residing in mount 'mnt'.
> + */
> +int seq_path(struct seq_file *m, struct vfsmount *mnt, struct dentry *dentry,
> + char *esc)
> {
> + char *p = NULL;
> + if (m->count < m->size) {
> + char *s = m->buf + m->count;
> - char *p = d_path(dentry, mnt, s, m->size - m->count);
> + p = d_path(dentry, mnt, s, m->size - m->count);
> + if (!IS_ERR(p)) {
> - while (s <= p) {
> - char c = *p++;
> - if (!c) {
> - p = m->buf + m->count;

```

```

> - m->count = s - m->buf;
> - return s - p;
> - } else if (!strchr(esc, c)) {
> - *s++ = c;
> - } else if (s + 4 > p) {
> - break;
> - } else {
> - *s++ = '\\';
> - *s++ = '0' + ((c & 0300) >> 6);
> - *s++ = '0' + ((c & 070) >> 3);
> - *s++ = '0' + (c & 07);
> - }
> + s = mangle_path(s, p, esc);
> + if (s) {
> + p = m->buf + m->count;
> + m->count = s - m->buf;
> + return s - p;
> }
> }
> }
> m->count = m->size;
> - return -1;
> + return p == ERR_PTR(-ENAMETOOLONG) ? 0 : -1;
> }
> +
> EXPORT_SYMBOL(seq_path);
>
> +/*
> + * returns the path of the 'dentry' from the root of its filesystem.
> + */
> +int seq_dentry(struct seq_file *m, struct dentry *dentry, char *esc)
> +{
> + char *p = NULL;
> + if (m->count < m->size) {
> + char *s = m->buf + m->count;
> + p = dentry_path(dentry, s, m->size - m->count);
> + if (!IS_ERR(p)) {
> + s = mangle_path(s, p, esc);
> + if (s) {
> + p = m->buf + m->count;
> + m->count = s - m->buf;
> + return s - p;
> + }
> + }
> + }
> + m->count = m->size;
> + return p == ERR_PTR(-ENAMETOOLONG) ? 0 : -1;
> +}

```

```

> +
> +EXPORT_SYMBOL(seq_dentry);
> +
> static void *single_start(struct seq_file *p, loff_t *pos)
> {
> return NULL + (*pos == 0);
> Index: linux-2.6.17.10/fs/proc/proc_misc.c
> =====
> --- linux-2.6.17.10.orig/fs/proc/proc_misc.c
> +++ linux-2.6.17.10/fs/proc/proc_misc.c
> @@ -675,6 +675,7 @@ void __init proc_misc_init(void)
> create_proc_read_entry(p->name, 0, NULL, p->read_proc, NULL);
>
> proc_symlink("mounts", NULL, "self/mounts");
> + proc_symlink("mounts_new", NULL, "self/mounts_new");
>
> /* And now for trickier ones */
> entry = create_proc_entry("kmsg", S_IRUSR, &proc_root);
> Index: linux-2.6.17.10/fs/namespace.c
> =====
> --- linux-2.6.17.10.orig/fs/namespace.c
> +++ linux-2.6.17.10/fs/namespace.c
> @@ -349,7 +349,7 @@ static inline void mangle(struct seq_fil
> seq_escape(m, s, "\t\n\\");
> }
>
> -static int show_vfsmnt(struct seq_file *m, void *v)
> +static int show_options(struct seq_file *m, void *v)
> {
> struct vfsmount *mnt = v;
> int err = 0;
> @@ -372,10 +372,6 @@ static int show_vfsmnt(struct seq_file *
> };
> struct proc_fs_info *fs_infol;
>
> - mangle(m, mnt->mnt_devname ? mnt->mnt_devname : "none");
> - seq_putc(m, ' ');
> - seq_path(m, mnt, mnt->mnt_root, "\t\n\\");
> - seq_putc(m, ' ');
> mangle(m, mnt->mnt_sb->s_type->name);
> seq_puts(m, mnt->mnt_sb->s_flags & MS_RDONLY ? " ro" : " rw");
> for (fs_infol = fs_info; fs_infol->flag; fs_infol++) {
> @@ -392,6 +388,28 @@ static int show_vfsmnt(struct seq_file *
> return err;
> }
>
> +static int show_vfsmnt(struct seq_file *m, void *v)
> +{

```

```

> + struct vfsmount *mnt = v;
> + mangle(m, mnt->mnt_devname ? mnt->mnt_devname : "none");
> + seq_putc(m, ' ');
> + seq_path(m, mnt, mnt->mnt_root, "\t\n\\");
> + seq_putc(m, ' ');
> + return show_options(m, v);
> +}
> +
> +static int show_vfsmnt_new(struct seq_file *m, void *v)
> +{
> + struct vfsmount *mnt = v;
> + seq_printf(m, "0x%x", new_encode_dev(mnt->mnt_sb->s_dev));
> + seq_putc(m, ' ');
> + seq_path(m, mnt, mnt->mnt_root, "\t\n\\");
> + seq_putc(m, ' ');
> + seq_dentry(m, mnt->mnt_root, "\t\n\\");
> + seq_putc(m, ' ');
> + return show_options(m, v);
> +}
> +
> struct seq_operations mounts_op = {
> .start = m_start,
> .next = m_next,
@@ -399,6 +417,13 @@ struct seq_operations mounts_op = {
> .show = show_vfsmnt
> };
>
> +struct seq_operations mounts_new_op = {
> + .start = m_start,
> + .next = m_next,
> + .stop = m_stop,
> + .show = show_vfsmnt_new
> +};
> +
> static int show_vfsstat(struct seq_file *m, void *v)
> {
> struct vfsmount *mnt = v;
> Index: linux-2.6.17.10/include/linux/dcache.h
> =====
> --- linux-2.6.17.10.orig/include/linux/dcache.h
> +++ linux-2.6.17.10/include/linux/dcache.h
@@ -281,6 +281,7 @@ extern struct dentry * d_hash_and_lookup
> extern int d_validate(struct dentry *, struct dentry *);
>
> extern char * d_path(struct dentry *, struct vfsmount *, char *, int);
> +extern char * dentry_path(struct dentry *, char *, int);
>
> /* Allocation counts.. */

```

```

>
> Index: linux-2.6.17.10/include/linux/seq_file.h
> =====
> --- linux-2.6.17.10.orig/include/linux/seq_file.h
> +++ linux-2.6.17.10/include/linux/seq_file.h
> @@ -43,6 +43,7 @@ int seq_printf(struct seq_file *, const
> __attribute__((format (printf,2,3)));
>
> int seq_path(struct seq_file *, struct vfsmount *, struct dentry *, char *);
> +int seq_dentry(struct seq_file *, struct dentry *, char *);
>
> int single_open(struct file *, int (*)(struct seq_file *, void *), void *);
> int single_release(struct inode *, struct file *);

> This patch introduces a new proc interface that exposes all the propagation
> trees within the namespace.
>
> It walks through each off the mounts in the namespace, and prints the following information.
>
> mount-id: a unique mount identifier
> dev-id : the unique device used to identify the device containing the filesystem
> path-from-root: mount point of the mount from /
> path-from-root-of-its-sb: path from its own root dentry.
> propagation-flag: SHARED, SLAVE, UNBINDABLE, PRIVATE
> peer-mount-id: the mount-id of its peer mount (if this mount is shared)
> master-mount-id: the mount-id of its master mount (if this mount is slave)
>
> Using the above information one could easily write a script that can
> draw all the propagation trees in the namespace.
>
>
> Example:
> Here is a sample output of cat /proc/$$/mounts_propagation
>
> 0xa917800 0x1 // PRIVATE
> 0xa917200 0x6200 // PRIVATE
> 0xa917180 0x3 /proc / PRIVATE
> 0xa917f80 0xa /dev/pts / PRIVATE
> 0xa917100 0x6210 /mnt / SHARED peer:0xa917100
> 0xa917f00 0x6210 /tmp /1 SLAVE master:0xa917100
> 0xa917900 0x6220 /mnt/2 / SHARED peer:0xa917900
>
> line 5 indicates that the mount with id 0xa917100 is mounted at /mnt is shared
> and it is the only mount in its peer group.
>
> line 6 indicates that the mount with id 0xa917f00 is mounted at /tmp, its
> root is the dentry 1 present under its root directory. This mount is a
> slave mount and its master is the mount with id 0xa917100.

```

```

>
> line 7 indicates that the mount with id 0xa917900 is mounted at /mnt/2, its
> root is the dentry / of its filesystem. This mount is a
> shared and it is the only mount in its peer group.
>
> one could write a script which runs through these lines and draws 4
> individual satellite mounts and two propagation trees, the first propagation
> tree has a shared mount and a slave mount. and the second propagation tree has
> just one shared mount.
>
>
> Signed-off-by: Ram Pai <linuxram@us.ibm.com>
> ---
> fs/namespace.c | 42 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
> fs/pnode.c     | 6 -----
> fs/pnode.h     | 6 ++++++
> fs/proc/base.c | 22 +++++++++++++++++++++++++++++++++-----
> 4 files changed, 69 insertions(+), 7 deletions(-)
>
> Index: linux-2.6.17.10/fs/namespace.c
> =====
> --- linux-2.6.17.10.orig/fs/namespace.c
> +++ linux-2.6.17.10/fs/namespace.c
> @@ -410,6 +410,41 @@ static int show_vfsmnt_new(struct seq_file
> return show_options(m, v);
> }
>
> +static int show_vfsmnt_propagation(struct seq_file *m, void *v)
> +{
> + struct vfsmount *mnt = v;
> + seq_printf(m, "0x%x", (int)mnt);
> + seq_putc(m, ' ');
> + seq_printf(m, "0x%x", new_encode_dev(mnt->mnt_sb->s_dev));
> + seq_putc(m, ' ');
> + seq_path(m, mnt, mnt->mnt_root, "\t\n\\");
> + seq_putc(m, ' ');
> + seq_dentry(m, mnt->mnt_root, "\t\n\\");
> + seq_putc(m, ' ');
> +
> + if (IS_MNT_SHARED(mnt)) {
> + seq_printf(m, "%s ", "SHARED");
> + if (IS_MNT_SLAVE(mnt)) {
> + seq_printf(m, "%s ", "SLAVE");
> + }
> + } else if (IS_MNT_SLAVE(mnt)) {
> + seq_printf(m, "%s ", "SLAVE");
> + } else if (IS_MNT_UNBINDABLE(mnt)) {
> + seq_printf(m, "%s ", "UNBINDABLE");

```

```
> + } else {
> + seq_printf(m, "%s ", "PRIVATE");
> + }
> +
> + if (IS_MNT_SHARED(mnt)) {
> + seq_printf(m, "peer:0x%x ", (int)next_peer(mnt));
```

Ok, so if the sequence of events was

```
mount --make-shared /mnt
(some user logs in and gets a cloned namespace, so his /mnt
becomes the next peer of /mnt)
mount --bind /mnt /tmp
(some other user logs in and gets cloned namespace...)
```

or some such sequence of events, we could lose all information about /mnt and /tmp being peers, right? Should a new `next_peer_in_same_namespace(mnt)` be used rather than `next_peer()`?

Somewhat similarly,

```
> + }
> + if (IS_MNT_SLAVE(mnt)) {
> + seq_printf(m, "master:0x%x ", (int)mnt->mnt_master);
```

Should we for privacy reasons not print out the address `mnt->mnt_master` is in a different namespace (perhaps if `!CAP_SYS_ADMIN`)?

Otherwise I like this.

thanks,
-serge

```
> + }
> + seq_puts(m, "\n");
> + return 0;
> +}
> +
> struct seq_operations mounts_op = {
> .start = m_start,
> .next = m_next,
> @@ -424,6 +459,13 @@ struct seq_operations mounts_new_op = {
> .show = show_vfsmnt_new
> };
>
> +struct seq_operations mounts_propagation_op = {
> + .start = m_start,
> + .next = m_next,
```

```

> + .stop = m_stop,
> + .show = show_vfsmnt_propagation
> +};
> +
> static int show_vfsstat(struct seq_file *m, void *v)
> {
>     struct vfsmount *mnt = v;
> Index: linux-2.6.17.10/fs/proc/base.c
> =====
> --- linux-2.6.17.10.orig/fs/proc/base.c
> +++ linux-2.6.17.10/fs/proc/base.c
> @@ -105,6 +105,7 @@ enum pid_directory_inos {
>     PROC_TGID_NUMA_MAPS,
>     PROC_TGID_MOUNTS,
>     PROC_TGID_MOUNTS_NEW,
> + PROC_TGID_MOUNTS_PROPAGATION,
>     PROC_TGID_MOUNTSTATS,
>     PROC_TGID_WCHAN,
> #ifdef CONFIG_MMU
> @@ -146,6 +147,7 @@ enum pid_directory_inos {
>     PROC_TID_MAPS,
>     PROC_TID_NUMA_MAPS,
>     PROC_TID_MOUNTS,
> + PROC_TID_MOUNTS_PROPAGATION,
>     PROC_TID_MOUNTS_NEW,
>     PROC_TID_MOUNTSTATS,
>     PROC_TID_WCHAN,
> @@ -206,6 +208,7 @@ static struct pid_entry tgid_base_stuff[
>     E(PROC_TGID_EXE, "exe", S_IFLNK|S_IRWXUGO),
>     E(PROC_TGID_MOUNTS, "mounts", S_IFREG|S_IRUGO),
>     E(PROC_TGID_MOUNTS_NEW,"mounts_new", S_IFREG|S_IRUGO),
> + E(PROC_TGID_MOUNTS_PROPAGATION,"mounts_propagation", S_IFREG|S_IRUGO),
>     E(PROC_TGID_MOUNTSTATS, "mountstats", S_IFREG|S_IRUSR),
> #ifdef CONFIG_MMU
>     E(PROC_TGID_SMAPS, "smaps", S_IFREG|S_IRUGO),
> @@ -250,6 +253,7 @@ static struct pid_entry tid_base_stuff[]
>     E(PROC_TID_EXE, "exe", S_IFLNK|S_IRWXUGO),
>     E(PROC_TID_MOUNTS, "mounts", S_IFREG|S_IRUGO),
>     E(PROC_TID_MOUNTS_NEW, "mounts_new", S_IFREG|S_IRUGO),
> + E(PROC_TID_MOUNTS_PROPAGATION,"mounts_propagation", S_IFREG|S_IRUGO),
> #ifdef CONFIG_MMU
>     E(PROC_TID_SMAPS, "smaps", S_IFREG|S_IRUGO),
> #endif
> @@ -733,7 +737,7 @@ static int __mounts_open(struct inode *i
>     return ret;
> }
>
> -extern struct seq_operations mounts_op, mounts_new_op;

```

```

> +extern struct seq_operations mounts_op, mounts_new_op, mounts_propagation_op;
> static int mounts_open(struct inode *inode, struct file *file)
> {
> return (__mounts_open(inode, file, &mounts_op));
> @@ -742,6 +746,10 @@ static int mounts_new_open(struct inode
> {
> return __mounts_open(inode, file, &mounts_new_op);
> }
> +static int mounts_propagation_open(struct inode *inode, struct file *file)
> +{
> + return __mounts_open(inode, file, &mounts_propagation_op);
> +}
>
> static int mounts_release(struct inode *inode, struct file *file)
> {
> @@ -785,6 +793,14 @@ static struct file_operations proc_mount
> .poll = mounts_poll,
> };
>
> +static struct file_operations proc_propagation_operations = {
> + .open = mounts_propagation_open,
> + .read = seq_read,
> + .llseek = seq_lseek,
> + .release = mounts_release,
> + .poll = mounts_poll,
> +};
> +
> extern struct seq_operations mountstats_op;
> static int mountstats_open(struct inode *inode, struct file *file)
> {
> @@ -1825,6 +1841,10 @@ static struct dentry *proc_pident_lookup
> case PROC_TGID_MOUNTS_NEW:
> inode->i_fop = &proc_mounts_new_operations;
> break;
> + case PROC_TID_MOUNTS_PROPAGATION:
> + case PROC_TGID_MOUNTS_PROPAGATION:
> + inode->i_fop = &proc_propagation_operations;
> + break;
> #ifdef CONFIG_MMU
> case PROC_TID_SMAPS:
> case PROC_TGID_SMAPS:
> Index: linux-2.6.17.10/fs/pnode.c
> =====
> --- linux-2.6.17.10.orig/fs/pnode.c
> +++ linux-2.6.17.10/fs/pnode.c
> @@ -11,12 +11,6 @@
> #include <linux/fs.h>
> #include "pnode.h"

```


> /dev/root /mnt ext2 rw 0 0
> /dev/root /tmp1 ext2 rw 0 0
>
> NOTE: The above mount entries, do not indicate that /tmp1 contains the same
> directory tree as /var/tmp.
>
> But 'cat /proc/mounts_new' shows us the following:
> 0x6200 /mnt /var ext2 rw 0 0
> 0x6200 /tmp1 /var/tmp ext2 rw 0 0

Can't you purely and simply add the fsid= option to /proc/mounts?

```
/dev/root /mnt ext2 rw,fsid=0x6200 0 0  
/dev/root /mnt ext2 rw,fsid=0x6200 0 0
```

I think you can do it without a negative impact to userspace.

> This patch introduces a new proc interface that exposes all the propagation
> trees within the namespace.

Good idea.

> It walks through each off the mounts in the namespace, and prints the following information.

>
> mount-id: a unique mount identifier
> dev-id : the unique device used to identify the device containing the filesystem
^^^

Why not major:minor?

> path-from-root: mount point of the mount from /
> path-from-root-of-its-sb: path from its own root dentry.
> propagation-flag: SHARED, SLAVE, UNBINDABLE, PRIVATE
> peer-mount-id: the mount-id of its peer mount (if this mount is shared)
> master-mount-id: the mount-id of its master mount (if this mount is slave)

> Example:

> Here is a sample output of cat /proc/\$\$/mounts_propagation

>
> 0xa917800 0x1 // PRIVATE
> 0xa917200 0x6200 // PRIVATE
> 0xa917180 0x3 /proc / PRIVATE
> 0xa917f80 0xa /dev/pts / PRIVATE
> 0xa917100 0x6210 /mnt / SHARED peer:0xa917100
> 0xa917f00 0x6210 /tmp /1 SLAVE master:0xa917100
> 0xa917900 0x6220 /mnt/2 / SHARED peer:0xa917900

Same thing (although the mounts_propagation makes more sense than
mount_new from my point of view).

cat /proc/mounts (or /proc/\$\$/mounts)

/dev/root /mnt ext2 rw,mid=0xa917100,did=0x6210,prop=SHARED,peer=0xa917100

my \$0.02...

Karel

--

Karel Zak <kzak@redhat.com>

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

Subject: Re: How to query mount propagation state?
Posted by [Ram Pai](#) on Tue, 17 Apr 2007 06:55:31 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Mon, 2007-04-16 at 14:16 -0500, Serge E. Hallyn wrote:

- > > This patch introduces a new proc interface that exposes all the
- > propagation
- > trees within the namespace.
- > >
- > > It walks through each off the mounts in the namespace, and prints
- > the following information.
- > >
- > > mount-id: a unique mount identifier
- > > dev-id : the unique device used to identify the device containing
- > the filesystem
- > > path-from-root: mount point of the mount from /
- > > path-from-root-of-its-sb: path from its own root dentry.
- > > propagation-flag: SHARED, SLAVE, UNBINDABLE, PRIVATE
- > > peer-mount-id: the mount-id of its peer mount (if this mount is
- > shared)
- > > master-mount-id: the mount-id of its master mount (if this mount is
- > slave)
- > >
- > > Using the above information one could easily write a script that can
- > > draw all the propagation trees in the namespace.
- > >
- > >
- > > Example:
- > > Here is a sample output of cat /proc/\$\$/mounts_propagation
- > >

```

> > 0xa917800 0x1 // PRIVATE
> > 0xa917200 0x6200 // PRIVATE
> > 0xa917180 0x3 /proc / PRIVATE
> > 0xa917f80 0xa /dev/pts / PRIVATE
> > 0xa917100 0x6210 /mnt / SHARED peer:0xa917100
> > 0xa917f00 0x6210 /tmp /1 SLAVE master:0xa917100
> > 0xa917900 0x6220 /mnt/2 / SHARED peer:0xa917900
> >
> > line 5 indicates that the mount with id 0xa917100 is mounted at /mnt
> > is shared
> > and it is the only mount in its peer group.
> >
> > line 6 indicates that the mount with id 0xa917f00 is mounted
> > at /tmp, its
> > root is the dentry 1 present under its root directory. This mount is
> > a
> > slave mount and its master is the mount with id 0xa917100.
> >
> > line 7 indicates that the mount with id 0xa917900 is mounted
> > at /mnt/2, its
> > root is the dentry / of its filesystem. This mount is a
> > shared and it is the only mount in its peer group.
> >
> > one could write a script which runs through these lines and draws 4
> > individual satellite mounts and two propagation trees, the first
> > propagation
> > tree has a shared mount and a slave mount. and the second
> > propagation tree has
> > just one shared mount.
> >
> >
> > Signed-off-by: Ram Pai <linuxram@us.ibm.com>
> > ---
> > fs/namespace.c | 42 ++++++
> > fs/pnode.c | 6 -----
> > fs/pnode.h | 6 ++++++
> > fs/proc/base.c | 22 ++++++
> > 4 files changed, 69 insertions(+), 7 deletions(-)
> >
> > Index: linux-2.6.17.10/fs/namespace.c
> > =====
> > --- linux-2.6.17.10.orig/fs/namespace.c
> > +++ linux-2.6.17.10/fs/namespace.c
> > @@ -410,6 +410,41 @@ static int show_vfsmnt_new(struct seq_fi
> >     return show_options(m, v);
> > }
> >
> > +static int show_vfsmnt_propagation(struct seq_file *m, void *v)

```

```

>>+{
>>+ struct vfsmount *mnt = v;
>>+ seq_printf(m, "0x%x", (int)mnt);
>>+ seq_putc(m, ' ');
>>+ seq_printf(m, "0x%x", new_encode_dev(mnt->mnt_sb->s_dev));
>>+ seq_putc(m, ' ');
>>+ seq_path(m, mnt, mnt->mnt_root, "\t\n\\");
>>+ seq_putc(m, ' ');
>>+ seq_dentry(m, mnt->mnt_root, "\t\n\\");
>>+ seq_putc(m, ' ');
>>+
>>+ if (IS_MNT_SHARED(mnt)) {
>>+     seq_printf(m, "%s ", "SHARED");
>>+     if (IS_MNT_SLAVE(mnt)) {
>>+         seq_printf(m, "%s ", "SLAVE");
>>+     }
>>+ } else if (IS_MNT_SLAVE(mnt)) {
>>+     seq_printf(m, "%s ", "SLAVE");
>>+ } else if (IS_MNT_UNBINDABLE(mnt)) {
>>+     seq_printf(m, "%s ", "UNBINDABLE");
>>+ } else {
>>+     seq_printf(m, "%s ", "PRIVATE");
>>+ }
>>+
>>+ if (IS_MNT_SHARED(mnt)) {
>>+     seq_printf(m, "peer:0x%x ", (int)next_peer(mnt));
>

```

> Ok, so if the sequence of events was

```

>
> mount --make-shared /mnt
> (some user logs in and gets a cloned namespace, so his /mnt
> becomes the next peer of /mnt)
> mount --bind /mnt /tmp
> (some other user logs in and gets cloned namespace...)
>

```

> or some such sequence of events, we could lose all information
> about /mnt and /tmp being peers, right? Should a new
> next_peer_in_same_namespace(mnt) be used rather than next_peer()?

you are right. it should print next_peer(mnt) only if CAP_SYS_ADMIN,
else print next_peer_in_same_namespace(mnt).

```

>
> Somewhat similarly,
>
>>+ }
>>+ if (IS_MNT_SLAVE(mnt)) {
>>+     seq_printf(m, "master:0x%x ", (int)mnt->mnt_master);

```

>
> Should we for privacy reasons not print out the address
> mnt->mnt_master
> is in a different namespace (perhaps if !CAP_SYS_ADMIN)?

right. it should print mnt->mnt_master if (CAP_SYS_ADMIN), otherwise
print master_in_same_namespace(mnt).

RP

>
> Otherwise I like this.
>
> thanks,
> -serge

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

Subject: Re: How to query mount propagation state?
Posted by [Ram Pai](#) on Tue, 17 Apr 2007 07:38:46 GMT
[View Forum Message](#) <> [Reply to Message](#)

On Mon, 2007-04-16 at 23:07 +0200, Karel Zak wrote:
> On Mon, Apr 16, 2007 at 10:39:46AM -0700, Ram Pai wrote:
>
> > This patch disambiguates multiple mount-instances of the same
> > filesystem (or part of the same filesystem), by introducing a new
> > interface /proc/mounts_new. The interface has the following format.
> ^^^^^^^^^^^^^^^
> ... odd name. What will be the name for a next generation?
> "/proc/mounts_new_new"? :-)
>

that was the name I came up with 6 months back :-). Yes It should be
something more appropriate. Maybe /proc/mounts_1 ? The next generation
one would be /proc/mounts_2 ? Suggestion?

> > 'cat /proc/mounts' shows the following:
> > /dev/root /mnt ext2 rw 0 0
> > /dev/root /tmp1 ext2 rw 0 0
> >
> > NOTE: The above mount entries, do not indicate that /tmp1 contains the same
> > directory tree as /var/tmp.

> >
> > But 'cat /proc/mounts_new' shows us the following:
> > 0x6200 /mnt /var ext2 rw 0 0
> > 0x6200 /tmp1 /var/tmp ext2 rw 0 0
>
> Can't you purely and simply add the fsid= option to /proc/mounts?
>
> /dev/root /mnt ext2 rw,fsid=0x6200 0 0
> /dev/root /mnt ext2 rw,fsid=0x6200 0 0
>
> I think you can do it without a negative impact to userspace.

ok.

>
> > This patch introduces a new proc interface that exposes all the propagation
> > trees within the namespace.
>
> Good idea.
>
> > It walks through each off the mounts in the namespace, and prints the following information.
> >
> > mount-id: a unique mount identifier
> > dev-id : the unique device used to identify the device containing the filesystem
> ^^^
> Why not major:minor?

Thinking about it, I feel we dont need this field at all. Basically we need a field that can be keyed-upon to find the corresponding record in /proc/mounts_1. mount-id can be used as the matching field, provided we add the mount-id field to /proc/mounts_1.
agree?

RP

> > path-from-root: mount point of the mount from /
> > path-from-root-of-its-sb: path from its own root dentry.
> > propagation-flag: SHARED, SLAVE, UNBINDABLE, PRIVATE
> > peer-mount-id: the mount-id of its peer mount (if this mount is shared)
> > master-mount-id: the mount-id of its master mount (if this mount is slave)
>
> > Example:
> > Here is a sample output of cat /proc/\$\$/mounts_propagation
> >
> > 0xa917800 0x1 // PRIVATE
> > 0xa917200 0x6200 // PRIVATE
> > 0xa917180 0x3 /proc / PRIVATE
> > 0xa917f80 0xa /dev/pts / PRIVATE

> > 0xa917100 0x6210 /mnt / SHARED peer:0xa917100
> > 0xa917f00 0x6210 /tmp /1 SLAVE master:0xa917100
> > 0xa917900 0x6220 /mnt/2 / SHARED peer:0xa917900
>
> Same thing (although the mounts_propagation makes more sense than
> mount_new from my point of view).
>
> cat /proc/mounts (or /proc/\$\$/mounts)
>
> /dev/root /mnt ext2 rw,mid=0xa917100,did=0x6210,prop=SHARED,peer=0xa917100
>
>
> my \$0.02...
>
> Karel
>

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