Subject: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Cedric Le Goater on Tue, 02 Oct 2007 08:46:09 GMT
View Forum Message <> Reply to Message

From: Cedric Le Goater <clg@fr.ibm.com>

This patch adds a struct mq_namespace holding the common attributes
of the mqueue namespace.

The current code is modified to use the default mqueue namespace
object 'init_mq_ns' and to prepare the ground for futur dynamic
objects.

Todo:
 - use CONFIG_NAMESPACE when next -mm is released

Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
---
 include/linux/mq_namespace.h | 60 ++++++++++++++++++++++
 ipc/mqueue.c | 111 ++++++++++++++++++++++++++++++++-----------
 2 files changed, 130 insertions(+), 41 deletions(-)

Index: 2.6.23-rc8-mm2/include/linux/mq_namespace.h
===
--- /dev/null
+++ 2.6.23-rc8-mm2/include/linux/mq_namespace.h
@@ -0,0 +1,60 @@
+#ifndef _LINUX_MQ_NAMESPACE_H
+#define _LINUX_MQ_NAMESPACE_H
+
+#include <linux/kref.h>
+
+struct vfsmount;
+
+struct mq_namespace {
+ struct kref kref;
+ struct vfsmount *mnt;
+
+ unsigned int queues_count;
+ unsigned int queues_max;
+ unsigned int msg_max;
+ unsigned int msgsize_max;
+};
+
+extern struct mq_namespace init_mq_ns;
+
+#ifdef CONFIG_POSIX_MQUEUE
+

```
+#define INIT_MQ_NS(ns)  .ns  = &init_mq_ns,
+
+static inline struct mq_namespace *get_mq_ns(struct mq_namespace *ns)
+{
+ if (ns)
+  kref_get(&ns->kref);
+ return ns;
+}
+
+extern struct mq_namespace *copy_mq_ns(unsigned long flags,
+     struct mq_namespace *old_ns);
+extern void free_mq_ns(struct kref *kref);
+
+static inline void put_mq_ns(struct mq_namespace *ns)
+{
+ if (ns)
+  kref_put(&ns->kref, free_mq_ns);
+}
+
+#else
+
+#define INIT_MQ_NS(ns)
+
+static inline struct mq_namespace *get_mq_ns(struct mq_namespace *ns)
+{
+ return ns;
+}
+
+static inline struct mq_namespace *copy_mq_ns(unsigned long flags,
+      struct mq_namespace *old_ns)
+{
+ return old_ns;
+}
+
+static inline void put_mq_ns(struct mq_namespace *ns) { }
+
+#endif /* CONFIG_POSIX_MQUEUE */
+
+#endif /* _LINUX_MQ_H */
Index: 2.6.23-rc8-mm2/ipc/mqueue.c
===================================================================
--- 2.6.23-rc8-mm2.orig/ipc/mqueue.c
+++ 2.6.23-rc8-mm2/ipc/mqueue.c
@@ -31,6 +31,7 @@
 #include <linux/mutex.h>
 #include <linux/nsproxy.h>
 #include <linux/pid.h>
+#include <linux/mq_namespace.h>
```

```
 #include <net/sock.h>
 #include "util.h"
@@ -87,12 +88,18 @@ static void remove_notification(struct m

 static spinlock_t mq_lock;
 static struct kmem_cache *mqueue_inode_cachep;
-static struct vfsmount *mqueue_mnt;

-static unsigned int queues_count;
-static unsigned int queues_max  = DFLT_QUEUESMAX;
-static unsigned int msg_max  = DFLT_MSGMAX;
-static unsigned int msgsize_max = DFLT_MSGSIZEMAX;
+struct mq_namespace init_mq_ns = {
+ .kref = {
+  .refcount = ATOMIC_INIT(2),
+ },
+ .mnt  = NULL,
+ .queues_count = 0,
+ .queues_max  = DFLT_QUEUESMAX,
+ .msg_max  = DFLT_MSGMAX,
+ .msgsize_max = DFLT_MSGSIZEMAX,
+};
+

 static struct ctl_table_header * mq_sysctl_table;

@@ -101,6 +108,21 @@ static inline struct mqueue_inode_info *
  return container_of(inode, struct mqueue_inode_info, vfs_inode);
 }

+struct mq_namespace *copy_mq_ns(unsigned long flags,
+    struct mq_namespace *old_ns)
+{
+ BUG_ON(!old_ns);
+ return get_mq_ns(old_ns);
+}
+
+void free_mq_ns(struct kref *kref)
+{
+ struct mq_namespace *mq_ns;
+
+ mq_ns = container_of(kref, struct mq_namespace, kref);
+ kfree(mq_ns);
+}
+
 static struct inode *mqueue_get_inode(struct super_block *sb, int mode,
        struct mq_attr *attr)
```

```
 {
@@ -235,6 +257,7 @@ static void mqueue_delete_inode(struct i
  struct user_struct *user;
  unsigned long mq_bytes;
  int i;
+ struct mq_namespace *mq_ns = &init_mq_ns;

  if (S_ISDIR(inode->i_mode)) {
   clear_inode(inode);
@@ -255,7 +278,7 @@ static void mqueue_delete_inode(struct i
  if (user) {
   spin_lock(&mq_lock);
   user->mq_bytes -= mq_bytes;
-  queues_count--;
+  mq_ns->queues_count--;
   spin_unlock(&mq_lock);
   free_uid(user);
  }
@@ -267,20 +290,22 @@ static int mqueue_create(struct inode *d
  struct inode *inode;
  struct mq_attr *attr = dentry->d_fsdata;
  int error;
+ struct mq_namespace *mq_ns = &init_mq_ns;

  spin_lock(&mq_lock);
- if (queues_count >= queues_max && !capable(CAP_SYS_RESOURCE)) {
+ if (mq_ns->queues_count >= mq_ns->queues_max &&
+  !capable(CAP_SYS_RESOURCE)) {
   error = -ENOSPC;
   goto out_lock;
  }
- queues_count++;
+ mq_ns->queues_count++;
  spin_unlock(&mq_lock);

  inode = mqueue_get_inode(dir->i_sb, mode, attr);
  if (!inode) {
   error = -ENOMEM;
   spin_lock(&mq_lock);
-  queues_count--;
+  mq_ns->queues_count--;
   goto out_lock;
  }

@@ -571,7 +596,7 @@ static void remove_notification(struct m
  info->notify_owner = NULL;
 }
```

```
-static int mq_attr_ok(struct mq_attr *attr)
+static int mq_attr_ok(struct mq_namespace *mq_ns, struct mq_attr *attr)
 {
  if (attr->mq_maxmsg <= 0 || attr->mq_msgsize <= 0)
   return 0;
@@ -579,8 +604,8 @@ static int mq_attr_ok(struct mq_attr *at
   if (attr->mq_maxmsg > HARD_MSGMAX)
    return 0;
  } else {
-  if (attr->mq_maxmsg > msg_max ||
-    attr->mq_msgsize > msgsize_max)
+  if (attr->mq_maxmsg > mq_ns->msg_max ||
+    attr->mq_msgsize > mq_ns->msgsize_max)
    return 0;
  }
  /* check for overflow */
@@ -596,8 +621,9 @@ static int mq_attr_ok(struct mq_attr *at
 /*
  * Invoked when creating a new queue via sys_mq_open
  */
-static struct file *do_create(struct dentry *dir, struct dentry *dentry,
-   int oflag, mode_t mode, struct mq_attr __user *u_attr)
+static struct file *do_create(struct mq_namespace *mq_ns, struct dentry *dir,
+   struct dentry *dentry, int oflag, mode_t mode,
+   struct mq_attr __user *u_attr)
 {
  struct mq_attr attr;
  int ret;
@@ -607,7 +633,7 @@ static struct file *do_create(struct den
   if (copy_from_user(&attr, u_attr, sizeof(attr)))
    goto out;
   ret = -EINVAL;
-  if (!mq_attr_ok(&attr))
+  if (!mq_attr_ok(mq_ns, &attr))
    goto out;
   /* store for use during create */
   dentry->d_fsdata = &attr;
@@ -619,33 +645,34 @@ static struct file *do_create(struct den
  if (ret)
   goto out;

- return dentry_open(dentry, mqueue_mnt, oflag);
+ return dentry_open(dentry, mq_ns->mnt, oflag);

 out:
  dput(dentry);
- mntput(mqueue_mnt);
+ mntput(mq_ns->mnt);
```

```
  return ERR_PTR(ret);
 }

 /* Opens existing queue */
-static struct file *do_open(struct dentry *dentry, int oflag)
+static struct file *do_open(struct mq_namespace *mq_ns, struct dentry *dentry,
+   int oflag)
 {
 static int oflag2acc[O_ACCMODE] = { MAY_READ, MAY_WRITE,
      MAY_READ | MAY_WRITE };

  if ((oflag & O_ACCMODE) == (O_RDWR | O_WRONLY)) {
   dput(dentry);
-  mntput(mqueue_mnt);
+  mntput(mq_ns->mnt);
   return ERR_PTR(-EINVAL);
  }

  if (permission(dentry->d_inode, oflag2acc[oflag & O_ACCMODE], NULL)) {
   dput(dentry);
-  mntput(mqueue_mnt);
+  mntput(mq_ns->mnt);
   return ERR_PTR(-EACCES);
  }

- return dentry_open(dentry, mqueue_mnt, oflag);
+ return dentry_open(dentry, mq_ns->mnt, oflag);
 }

 asmlinkage long sys_mq_open(const char __user *u_name, int oflag, mode_t mode,
@@ -655,6 +682,7 @@ asmlinkage long sys_mq_open(const char _
  struct file *filp;
  char *name;
  int fd, error;
+ struct mq_namespace *mq_ns = &init_mq_ns;

  error = audit_mq_open(oflag, mode, u_attr);
  if (error != 0)
@@ -667,13 +695,13 @@ asmlinkage long sys_mq_open(const char _
  if (fd < 0)
   goto out_putname;

- mutex_lock(&mqueue_mnt->mnt_root->d_inode->i_mutex);
- dentry = lookup_one_len(name, mqueue_mnt->mnt_root, strlen(name));
+ mutex_lock(&mq_ns->mnt->mnt_root->d_inode->i_mutex);
+ dentry = lookup_one_len(name, mq_ns->mnt->mnt_root, strlen(name));
  if (IS_ERR(dentry)) {
   error = PTR_ERR(dentry);
```

```
   goto out_err;
  }
- mntget(mqueue_mnt);
+ mntget(mq_ns->mnt);

  if (oflag & O_CREAT) {
   if (dentry->d_inode) { /* entry already exists */
@@ -681,12 +709,12 @@ asmlinkage long sys_mq_open(const char _
    error = -EEXIST;
    if (oflag & O_EXCL)
     goto out;
-   filp = do_open(dentry, oflag);
+   filp = do_open(mq_ns, dentry, oflag);
   } else {
-   error = mnt_want_write(mqueue_mnt);
+   error = mnt_want_write(mq_ns->mnt);
    if (error)
     goto out;
-   filp = do_create(mqueue_mnt->mnt_root, dentry,
+   filp = do_create(mq_ns, mq_ns->mnt->mnt_root, dentry,
       oflag, mode, u_attr);
   }
  } else {
@@ -694,7 +722,7 @@ asmlinkage long sys_mq_open(const char _
   if (!dentry->d_inode)
    goto out;
   audit_inode(name, dentry);
-  filp = do_open(dentry, oflag);
+  filp = do_open(mq_ns, dentry, oflag);
  }

  if (IS_ERR(filp)) {
@@ -708,13 +736,13 @@ asmlinkage long sys_mq_open(const char _

 out:
  dput(dentry);
- mntput(mqueue_mnt);
+ mntput(mq_ns->mnt);
 out_putfd:
  put_unused_fd(fd);
 out_err:
  fd = error;
 out_upsem:
- mutex_unlock(&mqueue_mnt->mnt_root->d_inode->i_mutex);
+ mutex_unlock(&mq_ns->mnt->mnt_root->d_inode->i_mutex);
 out_putname:
  putname(name);
  return fd;
```

```
@@ -726,14 +754,15 @@ asmlinkage long sys_mq_unlink(const char
  char *name;
  struct dentry *dentry;
  struct inode *inode = NULL;
+ struct mq_namespace *mq_ns = &init_mq_ns;

  name = getname(u_name);
  if (IS_ERR(name))
   return PTR_ERR(name);

- mutex_lock_nested(&mqueue_mnt->mnt_root->d_inode->i_mutex,
+ mutex_lock_nested(&mq_ns->mnt->mnt_root->d_inode->i_mutex,
    I_MUTEX_PARENT);
- dentry = lookup_one_len(name, mqueue_mnt->mnt_root, strlen(name));
+ dentry = lookup_one_len(name, mq_ns->mnt->mnt_root, strlen(name));
  if (IS_ERR(dentry)) {
   err = PTR_ERR(dentry);
   goto out_unlock;
@@ -747,16 +776,16 @@ asmlinkage long sys_mq_unlink(const char
  inode = dentry->d_inode;
  if (inode)
   atomic_inc(&inode->i_count);
- err = mnt_want_write(mqueue_mnt);
+ err = mnt_want_write(mq_ns->mnt);
  if (err)
   goto out_err;
  err = vfs_unlink(dentry->d_parent->d_inode, dentry);
- mnt_drop_write(mqueue_mnt);
+ mnt_drop_write(mq_ns->mnt);
 out_err:
  dput(dentry);

 out_unlock:
- mutex_unlock(&mqueue_mnt->mnt_root->d_inode->i_mutex);
+ mutex_unlock(&mq_ns->mnt->mnt_root->d_inode->i_mutex);
  putname(name);
  if (inode)
   iput(inode);
@@ -1201,14 +1230,14 @@ static int msg_maxsize_limit_max = INT_M
 static ctl_table mq_sysctls[] = {
  {
   .procname = "queues_max",
-  .data  = &queues_max,
+  .data  = &init_mq_ns.queues_max,
   .maxlen  = sizeof(int),
   .mode  = 0644,
   .proc_handler = &proc_dointvec,
  },
```

```
 {
  .procname = "msg_max",
-  .data  = &msg_max,
+  .data  = &init_mq_ns.msg_max,
  .maxlen  = sizeof(int),
  .mode  = 0644,
  .proc_handler = &proc_dointvec_minmax,
@@ -1217,7 +1246,7 @@ static ctl_table mq_sysctls[] = {
 },
 {
  .procname = "msgsize_max",
-  .data  = &msgsize_max,
+  .data  = &init_mq_ns.msgsize_max,
  .maxlen  = sizeof(int),
  .mode  = 0644,
  .proc_handler = &proc_dointvec_minmax,
@@ -1263,13 +1292,13 @@ static int __init init_mqueue_fs(void)
 if (error)
  goto out_sysctl;

- if (IS_ERR(mqueue_mnt = kern_mount(&mqueue_fs_type))) {
-  error = PTR_ERR(mqueue_mnt);
+ init_mq_ns.mnt = kern_mount(&mqueue_fs_type);
+ if (IS_ERR(init_mq_ns.mnt)) {
+  error = PTR_ERR(init_mq_ns.mnt);
  goto out_filesystem;
 }

 /* internal initialization - not common for vfs */
- queues_count = 0;
 spin_lock_init(&mq_lock);

 return 0;

--
```

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by dev on Tue, 02 Oct 2007 09:06:49 GMT
View Forum Message <> Reply to Message

Cedric,

how safe does it intersect with netlinks from network namespace?

I see mqueues can send netlink messages, have you checked how safe it is?

Thanks,
Kirill

Cedric Le Goater wrote:
> From: Cedric Le Goater <clg@fr.ibm.com>
>
> This patch adds a struct mq_namespace holding the common attributes
> of the mqueue namespace.
>
> The current code is modified to use the default mqueue namespace
> object 'init_mq_ns' and to prepare the ground for futur dynamic
> objects.
>
> Todo:
> - use CONFIG_NAMESPACE when next -mm is released
>
> Signed-off-by: Cedric Le Goater <clg@fr.ibm.com>
> ---
> include/linux/mq_namespace.h | 60 ++++++++++++++++++++++
> ipc/mqueue.c | 111 ++++++++++++++++++++++++++++++-----------
> 2 files changed, 130 insertions(+), 41 deletions(-)
>
> Index: 2.6.23-rc8-mm2/include/linux/mq_namespace.h
> ===
> --- /dev/null
> +++ 2.6.23-rc8-mm2/include/linux/mq_namespace.h
> @@ -0,0 +1,60 @@
> +#ifndef _LINUX_MQ_NAMESPACE_H
> +#define _LINUX_MQ_NAMESPACE_H
> +
> +#include <linux/kref.h>
> +
> +struct vfsmount;
> +
> +struct mq_namespace {
> + struct kref kref;
> + struct vfsmount *mnt;
> +
> + unsigned int queues_count;
> + unsigned int queues_max;
> + unsigned int msg_max;
> + unsigned int msgsize_max;
> +};
> +
> +extern struct mq_namespace init_mq_ns;
> +

```
> +#ifdef CONFIG_POSIX_MQUEUE
> +
> +#define INIT_MQ_NS(ns)  .ns  = &init_mq_ns,
> +
> +static inline struct mq_namespace *get_mq_ns(struct mq_namespace *ns)
> +{
> + if (ns)
> +  kref_get(&ns->kref);
> + return ns;
> +}
> +
> +extern struct mq_namespace *copy_mq_ns(unsigned long flags,
> +     struct mq_namespace *old_ns);
> +extern void free_mq_ns(struct kref *kref);
> +
> +static inline void put_mq_ns(struct mq_namespace *ns)
> +{
> + if (ns)
> +  kref_put(&ns->kref, free_mq_ns);
> +}
> +
> +#else
> +
> +#define INIT_MQ_NS(ns)
> +
> +static inline struct mq_namespace *get_mq_ns(struct mq_namespace *ns)
> +{
> + return ns;
> +}
> +
> +static inline struct mq_namespace *copy_mq_ns(unsigned long flags,
> +      struct mq_namespace *old_ns)
> +{
> + return old_ns;
> +}
> +
> +static inline void put_mq_ns(struct mq_namespace *ns) { }
> +
> +#endif /* CONFIG_POSIX_MQUEUE */
> +
> +#endif /* _LINUX_MQ_H */
> Index: 2.6.23-rc8-mm2/ipc/mqueue.c
> ===================================================================
> --- 2.6.23-rc8-mm2.orig/ipc/mqueue.c
> +++ 2.6.23-rc8-mm2/ipc/mqueue.c
> @@ -31,6 +31,7 @@
>  #include <linux/mutex.h>
>  #include <linux/nsproxy.h>
```

```
>  #include <linux/pid.h>
> +#include <linux/mq_namespace.h>
>
>  #include <net/sock.h>
>  #include "util.h"
> @@ -87,12 +88,18 @@ static void remove_notification(struct m
>
>  static spinlock_t mq_lock;
>  static struct kmem_cache *mqueue_inode_cachep;
> -static struct vfsmount *mqueue_mnt;
>
> -static unsigned int queues_count;
> -static unsigned int queues_max  = DFLT_QUEUESMAX;
> -static unsigned int msg_max  = DFLT_MSGMAX;
> -static unsigned int msgsize_max = DFLT_MSGSIZEMAX;
> +struct mq_namespace init_mq_ns = {
> + .kref = {
> +  .refcount = ATOMIC_INIT(2),
> + },
> + .mnt  = NULL,
> + .queues_count = 0,
> + .queues_max  = DFLT_QUEUESMAX,
> + .msg_max  = DFLT_MSGMAX,
> + .msgsize_max = DFLT_MSGSIZEMAX,
> +};
> +
>
>  static struct ctl_table_header * mq_sysctl_table;
>
> @@ -101,6 +108,21 @@ static inline struct mqueue_inode_info *
>   return container_of(inode, struct mqueue_inode_info, vfs_inode);
>  }
>
> +struct mq_namespace *copy_mq_ns(unsigned long flags,
> +    struct mq_namespace *old_ns)
> +{
> + BUG_ON(!old_ns);
> + return get_mq_ns(old_ns);
> +}
> +
> +void free_mq_ns(struct kref *kref)
> +{
> + struct mq_namespace *mq_ns;
> +
> + mq_ns = container_of(kref, struct mq_namespace, kref);
> + kfree(mq_ns);
> +}
> +
```

```
>  static struct inode *mqueue_get_inode(struct super_block *sb, int mode,
>         struct mq_attr *attr)
>  {
> @@ -235,6 +257,7 @@ static void mqueue_delete_inode(struct i
>   struct user_struct *user;
>   unsigned long mq_bytes;
>   int i;
> + struct mq_namespace *mq_ns = &init_mq_ns;
>
>   if (S_ISDIR(inode->i_mode)) {
>    clear_inode(inode);
> @@ -255,7 +278,7 @@ static void mqueue_delete_inode(struct i
>   if (user) {
>    spin_lock(&mq_lock);
>    user->mq_bytes -= mq_bytes;
> -  queues_count--;
> +  mq_ns->queues_count--;
>    spin_unlock(&mq_lock);
>    free_uid(user);
>   }
> @@ -267,20 +290,22 @@ static int mqueue_create(struct inode *d
>   struct inode *inode;
>   struct mq_attr *attr = dentry->d_fsdata;
>   int error;
> + struct mq_namespace *mq_ns = &init_mq_ns;
>
>   spin_lock(&mq_lock);
> - if (queues_count >= queues_max && !capable(CAP_SYS_RESOURCE)) {
> + if (mq_ns->queues_count >= mq_ns->queues_max &&
> +  !capable(CAP_SYS_RESOURCE)) {
>    error = -ENOSPC;
>    goto out_lock;
>   }
> - queues_count++;
> + mq_ns->queues_count++;
>   spin_unlock(&mq_lock);
>
>   inode = mqueue_get_inode(dir->i_sb, mode, attr);
>   if (!inode) {
>    error = -ENOMEM;
>    spin_lock(&mq_lock);
> -  queues_count--;
> +  mq_ns->queues_count--;
>    goto out_lock;
>   }
>
> @@ -571,7 +596,7 @@ static void remove_notification(struct m
>   info->notify_owner = NULL;
```

```
>  }
>
> -static int mq_attr_ok(struct mq_attr *attr)
> +static int mq_attr_ok(struct mq_namespace *mq_ns, struct mq_attr *attr)
>  {
>   if (attr->mq_maxmsg <= 0 || attr->mq_msgsize <= 0)
>    return 0;
> @@ -579,8 +604,8 @@ static int mq_attr_ok(struct mq_attr *at
>    if (attr->mq_maxmsg > HARD_MSGMAX)
>     return 0;
>   } else {
> -  if (attr->mq_maxmsg > msg_max ||
> -    attr->mq_msgsize > msgsize_max)
> +  if (attr->mq_maxmsg > mq_ns->msg_max ||
> +    attr->mq_msgsize > mq_ns->msgsize_max)
>     return 0;
>   }
>   /* check for overflow */
> @@ -596,8 +621,9 @@ static int mq_attr_ok(struct mq_attr *at
>  /*
>   * Invoked when creating a new queue via sys_mq_open
>   */
> -static struct file *do_create(struct dentry *dir, struct dentry *dentry,
> -   int oflag, mode_t mode, struct mq_attr __user *u_attr)
> +static struct file *do_create(struct mq_namespace *mq_ns, struct dentry *dir,
> +   struct dentry *dentry, int oflag, mode_t mode,
> +   struct mq_attr __user *u_attr)
>  {
>   struct mq_attr attr;
>   int ret;
> @@ -607,7 +633,7 @@ static struct file *do_create(struct den
>    if (copy_from_user(&attr, u_attr, sizeof(attr)))
>     goto out;
>    ret = -EINVAL;
> -  if (!mq_attr_ok(&attr))
> +  if (!mq_attr_ok(mq_ns, &attr))
>     goto out;
>    /* store for use during create */
>    dentry->d_fsdata = &attr;
> @@ -619,33 +645,34 @@ static struct file *do_create(struct den
>   if (ret)
>    goto out;
>
> - return dentry_open(dentry, mqueue_mnt, oflag);
> + return dentry_open(dentry, mq_ns->mnt, oflag);
>
>  out:
>   dput(dentry);
```

```
> - mntput(mqueue_mnt);
> + mntput(mq_ns->mnt);
>   return ERR_PTR(ret);
>  }
>
>  /* Opens existing queue */
> -static struct file *do_open(struct dentry *dentry, int oflag)
> +static struct file *do_open(struct mq_namespace *mq_ns, struct dentry *dentry,
> +    int oflag)
>  {
>  static int oflag2acc[O_ACCMODE] = { MAY_READ, MAY_WRITE,
>       MAY_READ | MAY_WRITE };
>
>   if ((oflag & O_ACCMODE) == (O_RDWR | O_WRONLY)) {
>    dput(dentry);
> -  mntput(mqueue_mnt);
> +  mntput(mq_ns->mnt);
>    return ERR_PTR(-EINVAL);
>   }
>
>   if (permission(dentry->d_inode, oflag2acc[oflag & O_ACCMODE], NULL)) {
>    dput(dentry);
> -  mntput(mqueue_mnt);
> +  mntput(mq_ns->mnt);
>    return ERR_PTR(-EACCES);
>   }
>
> - return dentry_open(dentry, mqueue_mnt, oflag);
> + return dentry_open(dentry, mq_ns->mnt, oflag);
>  }
>
>  asmlinkage long sys_mq_open(const char __user *u_name, int oflag, mode_t mode,
> @@ -655,6 +682,7 @@ asmlinkage long sys_mq_open(const char _
>   struct file *filp;
>   char *name;
>   int fd, error;
> + struct mq_namespace *mq_ns = &init_mq_ns;
>
>   error = audit_mq_open(oflag, mode, u_attr);
>   if (error != 0)
> @@ -667,13 +695,13 @@ asmlinkage long sys_mq_open(const char _
>   if (fd < 0)
>    goto out_putname;
>
> - mutex_lock(&mqueue_mnt->mnt_root->d_inode->i_mutex);
> - dentry = lookup_one_len(name, mqueue_mnt->mnt_root, strlen(name));
> + mutex_lock(&mq_ns->mnt->mnt_root->d_inode->i_mutex);
> + dentry = lookup_one_len(name, mq_ns->mnt->mnt_root, strlen(name));
```

```
>   if (IS_ERR(dentry)) {
>    error = PTR_ERR(dentry);
>    goto out_err;
>   }
> - mntget(mqueue_mnt);
> + mntget(mq_ns->mnt);
>
>   if (oflag & O_CREAT) {
>    if (dentry->d_inode) { /* entry already exists */
> @@ -681,12 +709,12 @@ asmlinkage long sys_mq_open(const char _
>     error = -EEXIST;
>     if (oflag & O_EXCL)
>      goto out;
> -   filp = do_open(dentry, oflag);
> +   filp = do_open(mq_ns, dentry, oflag);
>    } else {
> -   error = mnt_want_write(mqueue_mnt);
> +   error = mnt_want_write(mq_ns->mnt);
>     if (error)
>      goto out;
> -   filp = do_create(mqueue_mnt->mnt_root, dentry,
> +   filp = do_create(mq_ns, mq_ns->mnt->mnt_root, dentry,
>        oflag, mode, u_attr);
>    }
>   } else {
> @@ -694,7 +722,7 @@ asmlinkage long sys_mq_open(const char _
>    if (!dentry->d_inode)
>     goto out;
>    audit_inode(name, dentry);
> -  filp = do_open(dentry, oflag);
> +  filp = do_open(mq_ns, dentry, oflag);
>   }
>
>   if (IS_ERR(filp)) {
> @@ -708,13 +736,13 @@ asmlinkage long sys_mq_open(const char _
>
>  out:
>   dput(dentry);
> - mntput(mqueue_mnt);
> + mntput(mq_ns->mnt);
>  out_putfd:
>   put_unused_fd(fd);
>  out_err:
>   fd = error;
>  out_upsem:
> - mutex_unlock(&mqueue_mnt->mnt_root->d_inode->i_mutex);
> + mutex_unlock(&mq_ns->mnt->mnt_root->d_inode->i_mutex);
>  out_putname:
```

```
>   putname(name);
>   return fd;
> @@ -726,14 +754,15 @@ asmlinkage long sys_mq_unlink(const char
>   char *name;
>   struct dentry *dentry;
>   struct inode *inode = NULL;
> + struct mq_namespace *mq_ns = &init_mq_ns;
>
>   name = getname(u_name);
>   if (IS_ERR(name))
>    return PTR_ERR(name);
>
> - mutex_lock_nested(&mqueue_mnt->mnt_root->d_inode->i_mutex,
> + mutex_lock_nested(&mq_ns->mnt->mnt_root->d_inode->i_mutex,
>     I_MUTEX_PARENT);
> - dentry = lookup_one_len(name, mqueue_mnt->mnt_root, strlen(name));
> + dentry = lookup_one_len(name, mq_ns->mnt->mnt_root, strlen(name));
>   if (IS_ERR(dentry)) {
>    err = PTR_ERR(dentry);
>    goto out_unlock;
> @@ -747,16 +776,16 @@ asmlinkage long sys_mq_unlink(const char
>   inode = dentry->d_inode;
>   if (inode)
>    atomic_inc(&inode->i_count);
> - err = mnt_want_write(mqueue_mnt);
> + err = mnt_want_write(mq_ns->mnt);
>   if (err)
>    goto out_err;
>   err = vfs_unlink(dentry->d_parent->d_inode, dentry);
> - mnt_drop_write(mqueue_mnt);
> + mnt_drop_write(mq_ns->mnt);
>  out_err:
>   dput(dentry);
>
>  out_unlock:
> - mutex_unlock(&mqueue_mnt->mnt_root->d_inode->i_mutex);
> + mutex_unlock(&mq_ns->mnt->mnt_root->d_inode->i_mutex);
>   putname(name);
>   if (inode)
>    iput(inode);
> @@ -1201,14 +1230,14 @@ static int msg_maxsize_limit_max = INT_M
>  static ctl_table mq_sysctls[] = {
>   {
>    .procname = "queues_max",
> -  .data  = &queues_max,
> +  .data  = &init_mq_ns.queues_max,
>    .maxlen  = sizeof(int),
>    .mode  = 0644,
```

```
>    .proc_handler = &proc_dointvec,
>   },
>   {
>    .procname = "msg_max",
> -  .data  = &msg_max,
> +  .data  = &init_mq_ns.msg_max,
>    .maxlen  = sizeof(int),
>    .mode  = 0644,
>    .proc_handler = &proc_dointvec_minmax,
> @@ -1217,7 +1246,7 @@ static ctl_table mq_sysctls[] = {
>   },
>   {
>    .procname = "msgsize_max",
> -  .data  = &msgsize_max,
> +  .data  = &init_mq_ns.msgsize_max,
>    .maxlen  = sizeof(int),
>    .mode  = 0644,
>    .proc_handler = &proc_dointvec_minmax,
> @@ -1263,13 +1292,13 @@ static int __init init_mqueue_fs(void)
>   if (error)
>    goto out_sysctl;
>
> - if (IS_ERR(mqueue_mnt = kern_mount(&mqueue_fs_type))) {
> -  error = PTR_ERR(mqueue_mnt);
> + init_mq_ns.mnt = kern_mount(&mqueue_fs_type);
> + if (IS_ERR(init_mq_ns.mnt)) {
> +  error = PTR_ERR(init_mq_ns.mnt);
>    goto out_filesystem;
>   }
>
>   /* internal initialization - not common for vfs */
> - queues_count = 0;
>   spin_lock_init(&mq_lock);
>
>   return 0;
>
```

_____

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Cedric Le Goater on Tue, 02 Oct 2007 10:13:16 GMT
View Forum Message <> Reply to Message

Hello Kirill,

Kirill Korotaev wrote:
> Cedric,
>
> how safe does it intersect with netlinks from network namespace?
> I see mqueues can send netlink messages, have you checked how safe it is?

a ref is taken on the 'struct sock' in the mq_notify() syscall and the
skbuff which will be send to notify the user is also allocated in the
mq_notify() syscall. So we should be in the same net namespace when we
register the notification and when we notify.

I hope the net guys can confirm or we will easily check in the next
-lxc patchset which will merge this patchset with netns.

however, we have an issue with the signal notification in __do_notify()
we could kill a process in a different pid namespace.

C.

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by ebiederm on Tue, 02 Oct 2007 10:59:35 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> Hello Kirill,
>
> Kirill Korotaev wrote:
>> Cedric,
>>
>> how safe does it intersect with netlinks from network namespace?
>> I see mqueues can send netlink messages, have you checked how safe it is?
>
> a ref is taken on the 'struct sock' in the mq_notify() syscall and the
> skbuff which will be send to notify the user is also allocated in the
> mq_notify() syscall. So we should be in the same net namespace when we
> register the notification and when we notify.
>
> I hope the net guys can confirm or we will easily check in the next
> -lxc patchset which will merge this patchset with netns.
>

> however, we have an issue with the signal notification in __do_notify()
> we could kill a process in a different pid namespace.

So I took a quick look at the code as it is (before this patchset)
and the taking a reference to a socket and the taking a reference to
a struct pid should do the right thing when we intersect with other
namespaces. It certainly does not look like a fundamental issue.

In practice the patchset as written does conflict with the network
namespace work in the net-2.6.24 tree so some adjustments will need
to be made.

Eric

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Cedric Le Goater on Tue, 02 Oct 2007 12:21:27 GMT
View Forum Message <> Reply to Message

>> however, we have an issue with the signal notification in __do_notify()
>> we could kill a process in a different pid namespace.
>
> So I took a quick look at the code as it is (before this patchset)
> and the taking a reference to a socket and the taking a reference to
> a struct pid should do the right thing when we intersect with other
> namespaces. It certainly does not look like a fundamental issue.

right. this should be covered when the pid namespace signal handling is
complete. kill_pid_info() should fail to send a signal to a sibling or
a parent pid namespace.

I guess we should add a WARNING() to say that we're attempting to do so.

> In practice the patchset as written does conflict with the network
> namespace work in the net-2.6.24 tree so some adjustments will need
> to be made.

I think no more than fixing the CLONE flags in sched.h and the conflicts
in nsproxy.c.

Thanks !

C.

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Sukadev Bhattiprolu on Tue, 02 Oct 2007 16:30:20 GMT
View Forum Message <> Reply to Message

Cedric Le Goater [clg@fr.ibm.com] wrote:
|
| >> however, we have an issue with the signal notification in __do_notify()
| >> we could kill a process in a different pid namespace.
| >
| > So I took a quick look at the code as it is (before this patchset)
| > and the taking a reference to a socket and the taking a reference to
| > a struct pid should do the right thing when we intersect with other
| > namespaces. It certainly does not look like a fundamental issue.

|
| right. this should be covered when the pid namespace signal handling is
| complete. kill_pid_info() should fail to send a signal to a sibling or
| a parent pid namespace.
|
| I guess we should add a WARNING() to say that we're attempting to do so.

Just want to clarify how a signal is sent to a parent ns.

A process P1 sets itself up to be notified when a message arrives
on a queue.

P1 then clones P2 with CLONE_NEWPID.

P2 writes to the message queue and thus signals P1

What should the semantics be here ?

I guess it makes less sense for two namespaces to be dependent on the same
message queue this way. But, if P2 writes to the queue, technically, the
queue is not empty, so P1 should be notified, no ?

This sounds similar to the SIGIO signal case (F_SETOWN). My understanding
was that we would notify whoever was set to receive the notification, even
if they were in a parent ns (again my reasoning was its based on the state
of a file).

IOW, should we change kill_pid_info() ? If the caller can 'see' the
'struct pid' they can signal it. The expectation was that callers would
call find_vpid() and thus only see processes in their namespace.

|
| > In practice the patchset as written does conflict with the network
| > namespace work in the net-2.6.24 tree so some adjustments will need
| > to be made.
|
| I think no more than fixing the CLONE flags in sched.h and the conflicts
| in nsproxy.c.
|
| Thanks !
|
| C.

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by ebiederm on Tue, 02 Oct 2007 17:02:24 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

>>> however, we have an issue with the signal notification in __do_notify()
>>> we could kill a process in a different pid namespace.
>>
>> So I took a quick look at the code as it is (before this patchset)
>> and the taking a reference to a socket and the taking a reference to
>> a struct pid should do the right thing when we intersect with other
>> namespaces. It certainly does not look like a fundamental issue.
>
> right. this should be covered when the pid namespace signal handling is
> complete. kill_pid_info() should fail to send a signal to a sibling or
> a parent pid namespace.

Huh?

If we call sys_mq_notify and we become the owner then it should
not be a problem to send a signal to us.

> I guess we should add a WARNING() to say that we're attempting to do so.

I don't understand the problem that you are seeing.

Eric

_____

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by ebiederm on Tue, 02 Oct 2007 17:16:46 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com writes:

> Cedric Le Goater [clg@fr.ibm.com] wrote:
> |
> | >> however, we have an issue with the signal notification in __do_notify()
> | >> we could kill a process in a different pid namespace.
> | >
> | > So I took a quick look at the code as it is (before this patchset)
> | > and the taking a reference to a socket and the taking a reference to
> | > a struct pid should do the right thing when we intersect with other
> | > namespaces. It certainly does not look like a fundamental issue.
> >
> |
> | right. this should be covered when the pid namespace signal handling is
> | complete. kill_pid_info() should fail to send a signal to a sibling or
> | a parent pid namespace.
> |
> | I guess we should add a WARNING() to say that we're attempting to do so.
> >
> Just want to clarify how a signal is sent to a parent ns.
> >
> A process P1 sets itself up to be notified when a message arrives
> on a queue.
> >
> P1 then clones P2 with CLONE_NEWPID.
> >
> P2 writes to the message queue and thus signals P1
> >
> What should the semantics be here ?
> >
> I guess it makes less sense for two namespaces to be dependent on the same
> message queue this way. But, if P2 writes to the queue, technically, the
> queue is not empty, so P1 should be notified, no ?

Sounds right to me.

> This sounds similar to the SIGIO signal case (F_SETOWN). My understanding

> was that we would notify whoever was set to receive the notification, even
> if they were in a parent ns (again my reasoning was its based on the state
> of a file).

Yep.

> IOW, should we change kill_pid_info() ? If the caller can 'see' the
> 'struct pid' they can signal it. The expectation was that callers would
> call find_vpid() and thus only see processes in their namespace.

Ok. Now I'm concerned.

I deliberately designed the initial pid namespace infrastructure to allow
mixing like this. Because it is the right thing to do.

The expectation is that in general namespaces provide isolation simply
because you cannot see and thus cannot interact with other processes.
However isolation is not the purpose in life of namespaces and if you
use them in more creative ways mixing should work just fine. But
you have to use all of the namespaces together, and you have
to carefully set things up to guarantee isolation.

The really challenging case to handle here is what happens if we are
signaling to someone in a sibling pid namespace. What do we set the
parent pid in the siginfo struct to. I think we agreed that 0 (blame
the kernel) is the appropriate pid last time we talked about this.

I'm worried now that the concept of vpid has confused someone. It
still doesn't feel right to me to call one pid value more or less
virtual then any other so the concept of a virtual pid doesn't make
sense to me. The way I have always thought of it is:
- pid_nr(struct pid *)
  The pid in the current pid namespace.
- __pid_nr(struct pid_namespace, struct pid *)
  The pid in some specified pid namespace.

With struct pid being defined to be global and doing something
appropriate in all pid namespaces.

Thinking about this concern that Cedric raises is actually independent
of the mqueue namespace and seems to be totally a pid namespace thing.
Because the only way this happens if we happen to share the mqueue
namespace. (i.e. what we are doing now).

Eric

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Cedric Le Goater on Wed, 03 Oct 2007 07:12:56 GMT
View Forum Message <> Reply to Message

Eric W. Biederman wrote:
> sukadev@us.ibm.com writes:
>
>> Cedric Le Goater [clg@fr.ibm.com] wrote:
>> |
>> | >> however, we have an issue with the signal notification in __do_notify()
>> | >> we could kill a process in a different pid namespace.
>> | >
>> | > So I took a quick look at the code as it is (before this patchset)
>> | > and the taking a reference to a socket and the taking a reference to
>> | > a struct pid should do the right thing when we intersect with other
>> | > namespaces. It certainly does not look like a fundamental issue.
>>
>> |
>> | right. this should be covered when the pid namespace signal handling is
>> | complete. kill_pid_info() should fail to send a signal to a sibling or
>> | a parent pid namespace.
>> |
>> | I guess we should add a WARNING() to say that we're attempting to do so.
>>
>> Just want to clarify how a signal is sent to a parent ns.
>>
>> A process P1 sets itself up to be notified when a message arrives
>> on a queue.
>>
>> P1 then clones P2 with CLONE_NEWPID.
>>
>> P2 writes to the message queue and thus signals P1
>>
>> What should the semantics be here ?
>>
>> I guess it makes less sense for two namespaces to be dependent on the same
>> message queue this way. But, if P2 writes to the queue, technically, the
>> queue is not empty, so P1 should be notified, no ?
>
> Sounds right to me.

It's right for the mqueue namespace but wrong for the pid namespace because
we will possibly send a signal to a sibling pid namespace.

>> This sounds similar to the SIGIO signal case (F_SETOWN). My understanding
>> was that we would notify whoever was set to receive the notification, even
>> if they were in a parent ns (again my reasoning was its based on the state
>> of a file).
>
> Yep.
>
>> IOW, should we change kill_pid_info() ? If the caller can 'see' the
>> 'struct pid' they can signal it. The expectation was that callers would
>> call find_vpid() and thus only see processes in their namespace.
>
> Ok. Now I'm concerned.
>
> I deliberately designed the initial pid namespace infrastructure to allow
> mixing like this. Because it is the right thing to do.
>
> The expectation is that in general namespaces provide isolation simply
> because you cannot see and thus cannot interact with other processes.
> However isolation is not the purpose in life of namespaces and if you
> use them in more creative ways mixing should work just fine. But
> you have to use all of the namespaces together, and you have
> to carefully set things up to guarantee isolation.
>
> The really challenging case to handle here is what happens if we are
> signaling to someone in a sibling pid namespace. What do we set the
> parent pid in the siginfo struct to. I think we agreed that 0 (blame
> the kernel) is the appropriate pid last time we talked about this.

0 seems appropriate for signal coming from a parent namespace, yes. but
here we could be sending a signal from

> I'm worried now that the concept of vpid has confused someone. It
> still doesn't feel right to me to call one pid value more or less
> virtual then any other so the concept of a virtual pid doesn't make
> sense to me. The way I have always thought of it is:
> - pid_nr(struct pid *)
>   The pid in the current pid namespace.
> - __pid_nr(struct pid_namespace, struct pid *)
>   The pid in some specified pid namespace.
>
> With struct pid being defined to be global and doing something
> appropriate in all pid namespaces.
>
> Thinking about this concern that Cedric raises is actually independent
> of the mqueue namespace and seems to be totally a pid namespace thing.
> Because the only way this happens if we happen to share the mqueue
> namespace. (i.e. what we are doing now).

>
>
> Eric

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Cedric Le Goater on Wed, 03 Oct 2007 07:38:58 GMT
View Forum Message <> Reply to Message

[
I have big fingers this morning and I managed to send this email
while typing it ... see below for the end. I should be awake now :)
]

> The really challenging case to handle here is what happens if we are
> signaling to someone in a sibling pid namespace. What do we set the
> parent pid in the siginfo struct to. I think we agreed that 0 (blame
> the kernel) is the appropriate pid last time we talked about this.

0 seems appropriate for a signal coming from a parent namespace, yes. but
here, we could be sending a signal from a child or sibling namespace.

> I'm worried now that the concept of vpid has confused someone. It
> still doesn't feel right to me to call one pid value more or less
> virtual then any other so the concept of a virtual pid doesn't make
> sense to me. The way I have always thought of it is:
> - pid_nr(struct pid *)
> The pid in the current pid namespace.
> - __pid_nr(struct pid_namespace, struct pid *)
> The pid in some specified pid namespace.
>
> With struct pid being defined to be global and doing something
> appropriate in all pid namespaces.
>
> Thinking about this concern that Cedric raises is actually independent
> of the mqueue namespace and seems to be totally a pid namespace thing.
> Because the only way this happens if we happen to share the mqueue
> namespace. (i.e. what we are doing now).

Is there a way to catch this general issue (we have the same in sigio)
in the kill*(struct pid) routines ? spit a big warning when the
current->nsproxy->pid_ns is not a parent ?

Thanks !

C.

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Cedric Le Goater on Wed, 03 Oct 2007 07:44:40 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Cedric Le Goater [clg@fr.ibm.com] wrote:
> |
> | >> however, we have an issue with the signal notification in __do_notify()
> | >> we could kill a process in a different pid namespace.
> | >
> | > So I took a quick look at the code as it is (before this patchset)
> | > and the taking a reference to a socket and the taking a reference to
> | > a struct pid should do the right thing when we intersect with other
> | > namespaces. It certainly does not look like a fundamental issue.
> >
> | right. this should be covered when the pid namespace signal handling is
> | complete. kill_pid_info() should fail to send a signal to a sibling or
> | a parent pid namespace.
> |
> | I guess we should add a WARNING() to say that we're attempting to do so.
> >
> Just want to clarify how a signal is sent to a parent ns.
>
> A process P1 sets itself up to be notified when a message arrives
> on a queue.
>
> P1 then clones P2 with CLONE_NEWPID.
>
> P2 writes to the message queue and thus signals P1
>
> What should the semantics be here ?
>
> I guess it makes less sense for two namespaces to be dependent on the same
> message queue this way. But, if P2 writes to the queue, technically, the
> queue is not empty, so P1 should be notified, no ?
>
> This sounds similar to the SIGIO signal case (F_SETOWN). My understanding
> was that we would notify whoever was set to receive the notification, even

> if they were in a parent ns (again my reasoning was its based on the state
> of a file).

yes

> IOW, should we change kill_pid_info() ? If the caller can 'see' the
> 'struct pid' they can signal it. The expectation was that callers would
> call find_vpid() and thus only see processes in their namespace.

I think we have to decide on some limitations with signals and make sure
that we cannot send a signal to a sibling pid namespace. This can occur
in some special namespaces unshare configuration which should never be used
but to make sure, let's add a big WARNING when we detect such a pid namespace
violation.

If it is what you mean, I agree :)

Thanks,

C.

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by serue on Wed, 03 Oct 2007 13:59:55 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> sukadev@us.ibm.com wrote:
> > Cedric Le Goater [clg@fr.ibm.com] wrote:
> > |
> > | >> however, we have an issue with the signal notification in __do_notify()
> > | >> we could kill a process in a different pid namespace.
> > | >
> > | > So I took a quick look at the code as it is (before this patchset)
> > | > and the taking a reference to a socket and the taking a reference to
> > | > a struct pid should do the right thing when we intersect with other
> > | > namespaces. It certainly does not look like a fundamental issue.
> > >
> > |
> > | right. this should be covered when the pid namespace signal handling is
> > | complete. kill_pid_info() should fail to send a signal to a sibling or
> > | a parent pid namespace.
> > |
> > | I guess we should add a WARNING() to say that we're attempting to do so.

> >
> > Just want to clarify how a signal is sent to a parent ns.
> >
> > A process P1 sets itself up to be notified when a message arrives
> > on a queue.
> >
> > P1 then clones P2 with CLONE_NEWPID.
> >
> > P2 writes to the message queue and thus signals P1
> >
> > What should the semantics be here ?
> >
> > I guess it makes less sense for two namespaces to be dependent on the same
> > message queue this way. But, if P2 writes to the queue, technically, the
> > queue is not empty, so P1 should be notified, no ?
> >
> > This sounds similar to the SIGIO signal case (F_SETOWN). My understanding
> > was that we would notify whoever was set to receive the notification, even
> > if they were in a parent ns (again my reasoning was its based on the state
> > of a file).
>
> yes
>
> > IOW, should we change kill_pid_info() ? If the caller can 'see' the
> > 'struct pid' they can signal it. The expectation was that callers would
> > call find_vpid() and thus only see processes in their namespace.
>
> I think we have to decide on some limitations with signals

Yes we do, but

> and make sure
> that we cannot send a signal to a sibling pid namespace.

I think you and Eric (and I) are disagreeing about those limitations.
You take it for granted that a sibling pidns is off limits for signals.
But the signal wasn't sent using a pid, but using a file (in SIGIO
case). So since the fs was shared, the signal should be sent. An
event happened, and the receiver wants to know about it.

> This can occur
> in some special namespaces unshare configuration which should never be used
> but to make sure, let's add a big WARNING when we detect such a pid namespace
> violation.
>
> If it is what you mean, I agree :)
>
> Thanks,

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

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Cedric Le Goater on Wed, 03 Oct 2007 14:11:27 GMT
View Forum Message <> Reply to Message

> I think you and Eric (and I) are disagreeing about those limitations.
> You take it for granted that a sibling pidns is off limits for signals.
> But the signal wasn't sent using a pid, but using a file (in SIGIO
> case). So since the fs was shared, the signal should be sent. An
> event happened, and the receiver wants to know about it.

seen that way I agree.

si_code is set to SI_MESGQ, but what do we put in si_pid ? 0 ?

we could use the si_errno to pass extra info, like the sending process
lives in a // world ...

C.

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by ebiederm on Wed, 03 Oct 2007 14:32:19 GMT
View Forum Message <> Reply to Message

Cedric Le Goater <clg@fr.ibm.com> writes:

> [
> I have big fingers this morning and I managed to send this email
> while typing it ... see below for the end. I should be awake now :)
>]
>

>> The really challenging case to handle here is what happens if we are
>> signaling to someone in a sibling pid namespace. What do we set the
>> parent pid in the siginfo struct to. I think we agreed that 0 (blame
>> the kernel) is the appropriate pid last time we talked about this.
>
> 0 seems appropriate for a signal coming from a parent namespace, yes. but
> here, we could be sending a signal from a child or sibling namespace.

Hmm. I finally see the part of this that is problematic and
that we aren't currently handling.

> sig_i.si_signo = info->notify.sigev_signo;
> sig_i.si_errno = 0;
> sig_i.si_code = SI_MESGQ;
> sig_i.si_value = info->notify.sigev_value;
> sig_i.si_pid = current->tgid;
> sig_i.si_uid = current->uid;
>
> kill_pid_info(info->notify.sigev_signo, &sig_i, info->notify_owner);

The filling in of the signal info structure. My gut feel says that
should be a struct pid reference.   We need to be very careful with
the siginfo cases, and si_pid. Unless someone has built a mechanism
for dealing with this I haven't seen.

I still think the right approach here is that if the pid doesn't map
into the target processes pid namespace we should use 0.

I also strongly suspect that si_pid should be a struct pid and that we
should translate it in the receiving process if possible.

> Is there a way to catch this general issue (we have the same in sigio)
> in the kill*(struct pid) routines ? spit a big warning when the
> current->nsproxy->pid_ns is not a parent ?

The SIGIO case is even trickier, although that may count as always
coming from the kernel so this doesn't come up.

In both cases what is really happening a process in a sibling
pid namespace is doing something and the kernel is telling us about
it. So sending the signal is legitimate.

The difficulty is that the kernel can't express the process that did
something. From my perspective that appears to be a fundamental
limitation, and our only real choice is to send 0.

We have a similar limitation with the uid namespace as well in this

case.

We can implement this easily by passing a struct pid. And translating
just before we cross the kernel/user boundary. And if it doesn't
translate use 0. Basically this is just your check for seeing
if the destination pid namespace is the same or a parent of
the sending pid namespace. That is what I always envisioned
being in __pid_nr(). As I don't think it makes any sense to map
pids from sibling pid namespaces.

The reason we want in general to do the translation at the
destination process is because we may be sending a signal
to a process group which could have processes in multiple
pid namespaces.

Eric

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Sukadev Bhattiprolu on Wed, 03 Oct 2007 16:56:44 GMT
View Forum Message <> Reply to Message

Cedric Le Goater [clg@fr.ibm.com] wrote:
| > I think you and Eric (and I) are disagreeing about those limitations.
| > You take it for granted that a sibling pidns is off limits for signals.
| > But the signal wasn't sent using a pid, but using a file (in SIGIO
| > case). So since the fs was shared, the signal should be sent. An
| > event happened, and the receiver wants to know about it.
|
| seen that way I agree.
|
| si_code is set to SI_MESGQ, but what do we put in si_pid ? 0 ?
|
| we could use the si_errno to pass extra info, like the sending process
| lives in a // world ...

Does the receiver need to know that sender is in a // world ?

|
| C.

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by ebiederm on Wed, 03 Oct 2007 17:03:26 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com writes:

> Cedric Le Goater [clg@fr.ibm.com] wrote:
> | > I think you and Eric (and I) are disagreeing about those limitations.
> | > You take it for granted that a sibling pidns is off limits for signals.
> | > But the signal wasn't sent using a pid, but using a file (in SIGIO
> | > case). So since the fs was shared, the signal should be sent. An
> | > event happened, and the receiver wants to know about it.
> |
> | seen that way I agree.
> |
> | si_code is set to SI_MESGQ, but what do we put in si_pid ? 0 ?
> |
> | we could use the si_errno to pass extra info, like the sending process
> | lives in a // world ...
>
> Does the receiver need to know that sender is in a // world ?

What is a // world ?

Eric

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

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Sukadev Bhattiprolu on Wed, 03 Oct 2007 17:09:30 GMT
View Forum Message <> Reply to Message

Eric W. Biederman [ebiederm@xmission.com] wrote:
| sukadev@us.ibm.com writes:
|
| > Cedric Le Goater [clg@fr.ibm.com] wrote:
| > | > I think you and Eric (and I) are disagreeing about those limitations.
| > | > You take it for granted that a sibling pidns is off limits for signals.
| > | > But the signal wasn't sent using a pid, but using a file (in SIGIO
| > | > case). So since the fs was shared, the signal should be sent. An
| > | > event happened, and the receiver wants to know about it.
| > |
| > | seen that way I agree.
| > |
| > | si_code is set to SI_MESGQ, but what do we put in si_pid ? 0 ?

| > |
| > | we could use the si_errno to pass extra info, like the sending process
| > | lives in a // world ...
| >
| > Does the receiver need to know that sender is in a // world ?
|
| What is a // world ?

Parallel world/universe :-)

I am assuming Cedric used that to refer to a sibling pid ns.

|
| Eric

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by Cedric Le Goater on Thu, 04 Oct 2007 13:12:39 GMT
View Forum Message <> Reply to Message

sukadev@us.ibm.com wrote:
> Eric W. Biederman [ebiederm@xmission.com] wrote:
> | sukadev@us.ibm.com writes:
> |
> | > Cedric Le Goater [clg@fr.ibm.com] wrote:
> | > | > I think you and Eric (and I) are disagreeing about those limitations.
> | > | > You take it for granted that a sibling pidns is off limits for signals.
> | > | > But the signal wasn't sent using a pid, but using a file (in SIGIO
> | > | > case). So since the fs was shared, the signal should be sent. An
> | > | > event happened, and the receiver wants to know about it.
> | > |
> | > | seen that way I agree.
> | > |
> | > | si_code is set to SI_MESGQ, but what do we put in si_pid ? 0 ?
> | > |
> | > | we could use the si_errno to pass extra info, like the sending process
> | > | lives in a // world ...
> | >
> | > Does the receiver need to know that sender is in a // world ?

probably not. it would mean that the user is container aware. bad idea.

> | What is a // world ?
>

> Parallel world/universe :-)
>
> I am assuming Cedric used that to refer to a sibling pid ns.

yes :)

Thanks !

C.

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

---

Subject: Re: [patch -mm 1/5] mqueue namespace : add struct mq_namespace
Posted by serue on Thu, 04 Oct 2007 13:32:36 GMT
View Forum Message <> Reply to Message

Quoting Cedric Le Goater (clg@fr.ibm.com):
> sukadev@us.ibm.com wrote:
> > Eric W. Biederman [ebiederm@xmission.com] wrote:
> > | sukadev@us.ibm.com writes:
> > |
> > | > Cedric Le Goater [clg@fr.ibm.com] wrote:
> > | > | > I think you and Eric (and I) are disagreeing about those limitations.
> > | > | > You take it for granted that a sibling pidns is off limits for signals.
> > | > | > But the signal wasn't sent using a pid, but using a file (in SIGIO
> > | > | > case). So since the fs was shared, the signal should be sent. An
> > | > | > event happened, and the receiver wants to know about it.
> > | > |
> > | > | seen that way I agree.
> > | > |
> > | > | si_code is set to SI_MESGQ, but what do we put in si_pid ? 0 ?
> > | > |
> > | > | we could use the si_errno to pass extra info, like the sending process
> > | > | lives in a // world ...
> > | >
> > | > Does the receiver need to know that sender is in a // world ?
> >
> probably not. it would mean that the user is container aware. bad idea.

Remember we don't have to hide the fact that the user is in a
container. Just enough to make it convenient, but not to the
point of going out of our way to try and hide the fact for no
other reason than to hide the fact.

> > | What is a // world ?

> >
> > Parallel world/universe :-)
> >
> > I am assuming Cedric used that to refer to a sibling pid ns.
>
> yes :)
>
> Thanks !
>
> C.

_____

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