Subject: [PATCH 0/4] SUNRPC: several fixes around PipeFS objects
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 13:48:43 GMT
View Forum Message <> Reply to Message

First two pathes fixes lockdep warnings and next two - dereferencing of
released pipe data on eventfd close.

The following series consists of:

Stanislav Kinsbursky (4):
  SUNRPC: replace per-net client lock by rw mutex
  NFS: replace per-net client lock by mutex
  SUNRPC: check RPC inode's pipe reference before dereferencing
  SUNRPC: move waitq from RPC pipe to RPC inode

```
 fs/nfs/client.c                      |   36 +++++++++---------
 fs/nfs/idmap.c                       |    4 +-
 fs/nfs/netns.h                       |    2 +
 include/linux/sunrpc/rpc_pipe_fs.h |    2 +
 net/sunrpc/clnt.c                    |   16 ++++----
 net/sunrpc/netns.h                   |    2 +
 net/sunrpc/rpc_pipe.c                |   71 +++++++++++++++++++++++-------------
 net/sunrpc/sunrpc_syms.c             |    2 +
 8 files changed, 77 insertions(+), 58 deletions(-)
```

Subject: [PATCH 3/4] SUNRPC: check RPC inode's pipe reference before dereferencing
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 13:49:08 GMT
View Forum Message <> Reply to Message

There are 2 tightly bound objects: pipe data (created for kernel needs, has
reference to dentry, which depends on PipeFS mount/umount) and PipeFS
dentry/inode pair (created on mount for user-space needs). They both
independently may have or have not a valid reference to each other.
This means, that we have to make sure, that pipe->dentry reference is valid on
upcalls, and dentry->pipe reference is valid on downcalls. The latter check is
absent - my fault.
IOW, PipeFS dentry can be opened by some process (rpc.idmapd for example), but
it's pipe data can belong to NFS mount, which was unmounted already and thus
pipe data was destroyed.
To fix this, pipe reference have to be set to NULL on rpc_unlink() and checked
on PipeFS file operations instead of pipe->dentry check.

Note: PipeFS "poll" file operation will be updated in next patch, because it's

logic is more complicated.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>

---
 net/sunrpc/rpc_pipe.c | 32 ++++++++++++++++++++------------
 1 files changed, 19 insertions(+), 13 deletions(-)

```
diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index 6873c9b..b67b2ae 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -174,6 +174,7 @@ rpc_close_pipes(struct inode *inode)
    pipe->ops->release_pipe(inode);
   cancel_delayed_work_sync(&pipe->queue_timeout);
   rpc_inode_setowner(inode, NULL);
+ RPC_I(inode)->pipe = NULL;
   mutex_unlock(&inode->i_mutex);
 }

@@ -203,12 +204,13 @@ rpc_destroy_inode(struct inode *inode)
 static int
 rpc_pipe_open(struct inode *inode, struct file *filp)
 {
- struct rpc_pipe *pipe = RPC_I(inode)->pipe;
+ struct rpc_pipe *pipe;
   int first_open;
   int res = -ENXIO;

   mutex_lock(&inode->i_mutex);
- if (pipe->dentry == NULL)
+ pipe = RPC_I(inode)->pipe;
+ if (pipe == NULL)
    goto out;
   first_open = pipe->nreaders == 0 && pipe->nwriters == 0;
   if (first_open && pipe->ops->open_pipe) {
@@ -229,12 +231,13 @@ out:
 static int
 rpc_pipe_release(struct inode *inode, struct file *filp)
 {
- struct rpc_pipe *pipe = RPC_I(inode)->pipe;
+ struct rpc_pipe *pipe;
   struct rpc_pipe_msg *msg;
   int last_close;

   mutex_lock(&inode->i_mutex);
- if (pipe->dentry == NULL)
+ pipe = RPC_I(inode)->pipe;
```

```
+ if (pipe == NULL)
   goto out;
  msg = filp->private_data;
  if (msg != NULL) {
@@ -270,12 +273,13 @@ static ssize_t
 rpc_pipe_read(struct file *filp, char __user *buf, size_t len, loff_t *offset)
 {
  struct inode *inode = filp->f_path.dentry->d_inode;
- struct rpc_pipe *pipe = RPC_I(inode)->pipe;
+ struct rpc_pipe *pipe;
  struct rpc_pipe_msg *msg;
  int res = 0;

  mutex_lock(&inode->i_mutex);
- if (pipe->dentry == NULL) {
+ pipe = RPC_I(inode)->pipe;
+ if (pipe == NULL) {
   res = -EPIPE;
   goto out_unlock;
  }
@@ -313,13 +317,12 @@ static ssize_t
 rpc_pipe_write(struct file *filp, const char __user *buf, size_t len, loff_t *offset)
 {
  struct inode *inode = filp->f_path.dentry->d_inode;
- struct rpc_pipe *pipe = RPC_I(inode)->pipe;
  int res;

  mutex_lock(&inode->i_mutex);
  res = -EPIPE;
- if (pipe->dentry != NULL)
-  res = pipe->ops->downcall(filp, buf, len);
+ if (RPC_I(inode)->pipe != NULL)
+  res = RPC_I(inode)->pipe->ops->downcall(filp, buf, len);
  mutex_unlock(&inode->i_mutex);
  return res;
 }
@@ -344,16 +347,18 @@ static long
 rpc_pipe_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
 {
  struct inode *inode = filp->f_path.dentry->d_inode;
- struct rpc_pipe *pipe = RPC_I(inode)->pipe;
+ struct rpc_pipe *pipe;
  int len;

  switch (cmd) {
  case FIONREAD:
-  spin_lock(&pipe->lock);
-  if (pipe->dentry == NULL) {
```

```
-   spin_unlock(&pipe->lock);
+  mutex_lock(&inode->i_mutex);
+  pipe = RPC_I(inode)->pipe;
+  if (pipe == NULL) {
+   mutex_unlock(&inode->i_mutex);
    return -EPIPE;
   }
+  spin_lock(&pipe->lock);
   len = pipe->pipelen;
   if (filp->private_data) {
    struct rpc_pipe_msg *msg;
@@ -361,6 +366,7 @@ rpc_pipe_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
    len += msg->len - msg->copied;
   }
   spin_unlock(&pipe->lock);
+  mutex_unlock(&inode->i_mutex);
   return put_user(len, (int __user *)arg);
  default:
   return -EINVAL;
```

Subject: [PATCH 4/4] SUNRPC: move waitq from RPC pipe to RPC inode
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 13:49:16 GMT
View Forum Message <> Reply to Message

Currently, wait queue, used for polling of RPC pipe changes from user-space,
is a part of RPC pipe. But the pipe data itself can be released on NFS umount
prior to dentry-inode pair, connected to it (is case of this pair is open by
some process).
This is not a problem for almost all pipe users, because all PipeFS file
operations checks pipe reference prior to using it.
Except evenfd. This thing registers itself with "poll" file operation and thus
has a reference to pipe wait queue. This leads to oopses on destroying eventfd
after NFS umount (like rpc_idmapd do) since not pipe data left to the point
already.
The solution is to wait queue from pipe data to internal RPC inode data. This
looks more logical, because this wiat queue used only for user-space processes,
which already holds inode reference.

Note: upcalls have to get pipe->dentry prior to dereferecing wait queue to make
sure, that mount point won't disappear from underneath us.

Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>

```
---
 include/linux/sunrpc/rpc_pipe_fs.h |    2 +-
 net/sunrpc/rpc_pipe.c              |   39 +++++++++++++++++++++++++++------------
 2 files changed, 27 insertions(+), 14 deletions(-)
```

```
diff --git a/include/linux/sunrpc/rpc_pipe_fs.h b/include/linux/sunrpc/rpc_pipe_fs.h
index 426ce6e..a7b422b 100644
--- a/include/linux/sunrpc/rpc_pipe_fs.h
+++ b/include/linux/sunrpc/rpc_pipe_fs.h
@@ -28,7 +28,6 @@ struct rpc_pipe {
  int pipelen;
  int nreaders;
  int nwriters;
- wait_queue_head_t waitq;
 #define RPC_PIPE_WAIT_FOR_OPEN 1
  int flags;
  struct delayed_work queue_timeout;
@@ -41,6 +40,7 @@ struct rpc_inode {
  struct inode vfs_inode;
  void *private;
  struct rpc_pipe *pipe;
+ wait_queue_head_t waitq;
 };

 static inline struct rpc_inode *
diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index b67b2ae..ac9ee15 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -57,7 +57,7 @@ void rpc_pipefs_notifier_unregister(struct notifier_block *nb)
 }
 EXPORT_SYMBOL_GPL(rpc_pipefs_notifier_unregister);

-static void rpc_purge_list(struct rpc_pipe *pipe, struct list_head *head,
+static void rpc_purge_list(wait_queue_head_t *waitq, struct list_head *head,
   void (*destroy_msg)(struct rpc_pipe_msg *), int err)
 {
  struct rpc_pipe_msg *msg;
@@ -70,7 +70,7 @@ static void rpc_purge_list(struct rpc_pipe *pipe, struct list_head *head,
   msg->errno = err;
   destroy_msg(msg);
  } while (!list_empty(head));
- wake_up(&pipe->waitq);
+ wake_up(waitq);
 }

 static void
@@ -80,6 +80,7 @@ rpc_timeout_upcall_queue(struct work_struct *work)
  struct rpc_pipe *pipe =
   container_of(work, struct rpc_pipe, queue_timeout.work);
  void (*destroy_msg)(struct rpc_pipe_msg *);
+ struct dentry *dentry;
```

```
  spin_lock(&pipe->lock);
  destroy_msg = pipe->ops->destroy_msg;
@@ -87,8 +88,13 @@ rpc_timeout_upcall_queue(struct work_struct *work)
   list_splice_init(&pipe->pipe, &free_list);
   pipe->pipelen = 0;
  }
+ dentry = dget(pipe->dentry);
  spin_unlock(&pipe->lock);
- rpc_purge_list(pipe, &free_list, destroy_msg, -ETIMEDOUT);
+ if (dentry) {
+  rpc_purge_list(&RPC_I(dentry->d_inode)->waitq,
+         &free_list, destroy_msg, -ETIMEDOUT);
+  dput(dentry);
+ }
 }

 ssize_t rpc_pipe_generic_upcall(struct file *filp, struct rpc_pipe_msg *msg,
@@ -125,6 +131,7 @@ int
 rpc_queue_upcall(struct rpc_pipe *pipe, struct rpc_pipe_msg *msg)
 {
  int res = -EPIPE;
+ struct dentry *dentry;

  spin_lock(&pipe->lock);
  if (pipe->nreaders) {
@@ -140,8 +147,12 @@ rpc_queue_upcall(struct rpc_pipe *pipe, struct rpc_pipe_msg *msg)
   pipe->pipelen += msg->len;
   res = 0;
  }
+ dentry = dget(pipe->dentry);
  spin_unlock(&pipe->lock);
- wake_up(&pipe->waitq);
+ if (dentry) {
+  wake_up(&RPC_I(dentry->d_inode)->waitq);
+  dput(dentry);
+ }
  return res;
 }
 EXPORT_SYMBOL_GPL(rpc_queue_upcall);
@@ -168,7 +179,7 @@ rpc_close_pipes(struct inode *inode)
  pipe->pipelen = 0;
  pipe->dentry = NULL;
  spin_unlock(&pipe->lock);
- rpc_purge_list(pipe, &free_list, pipe->ops->destroy_msg, -EPIPE);
+ rpc_purge_list(&RPC_I(inode)->waitq, &free_list, pipe->ops->destroy_msg, -EPIPE);
  pipe->nwriters = 0;
  if (need_release && pipe->ops->release_pipe)
```

```
   pipe->ops->release_pipe(inode);
@@ -257,7 +268,7 @@ rpc_pipe_release(struct inode *inode, struct file *filp)
    list_splice_init(&pipe->pipe, &free_list);
    pipe->pipelen = 0;
    spin_unlock(&pipe->lock);
-   rpc_purge_list(pipe, &free_list,
+   rpc_purge_list(&RPC_I(inode)->waitq, &free_list,
      pipe->ops->destroy_msg, -EAGAIN);
   }
  }
@@ -330,16 +341,18 @@ rpc_pipe_write(struct file *filp, const char __user *buf, size_t len, loff_t
*of
 static unsigned int
 rpc_pipe_poll(struct file *filp, struct poll_table_struct *wait)
 {
- struct rpc_pipe *pipe = RPC_I(filp->f_path.dentry->d_inode)->pipe;
- unsigned int mask = 0;
+ struct inode *inode = filp->f_path.dentry->d_inode;
+ struct rpc_inode *rpci = RPC_I(inode);
+ unsigned int mask = POLLOUT | POLLWRNORM;

- poll_wait(filp, &pipe->waitq, wait);
+ poll_wait(filp, &rpci->waitq, wait);

- mask = POLLOUT | POLLWRNORM;
- if (pipe->dentry == NULL)
+ mutex_lock(&inode->i_mutex);
+ if (rpci->pipe == NULL)
   mask |= POLLERR | POLLHUP;
- if (filp->private_data || !list_empty(&pipe->pipe))
+ else if (filp->private_data || !list_empty(&rpci->pipe->pipe))
   mask |= POLLIN | POLLRDNORM;
+ mutex_unlock(&inode->i_mutex);
  return mask;
 }

@@ -543,7 +556,6 @@ init_pipe(struct rpc_pipe *pipe)
  INIT_LIST_HEAD(&pipe->in_downcall);
  INIT_LIST_HEAD(&pipe->pipe);
  pipe->pipelen = 0;
- init_waitqueue_head(&pipe->waitq);
  INIT_DELAYED_WORK(&pipe->queue_timeout,
        rpc_timeout_upcall_queue);
  pipe->ops = NULL;
@@ -1165,6 +1177,7 @@ init_once(void *foo)
  inode_init_once(&rpci->vfs_inode);
  rpci->private = NULL;
  rpci->pipe = NULL;
```

```
+ init_waitqueue_head(&rpci->waitq);
 }

 int register_rpc_pipefs(void)
```