Subject: [PATCH v3 4/4] SUNRPC: move waitq from RPC pipe to RPC inode
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 18:05:54 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)
```