
Subject: [PATCH 3/6] SUNRPC: cleanup PipeFS redundant RPC inode usage
Posted by [Stanislav Kinsbursky](#) on Tue, 22 Nov 2011 14:45:37 GMT
[View Forum Message](#) <> [Reply to Message](#)

This patch removes redundant RPC inode references from PipeFS. These places are actually where pipes operations are performed.

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

```
net/sunrpc/rpc_pipe.c | 93 ++++++-----  
1 files changed, 46 insertions(+), 47 deletions(-)
```

```
diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c  
index a95ba18..edf140a 100644
```

```
--- a/net/sunrpc/rpc_pipe.c  
+++ b/net/sunrpc/rpc_pipe.c  
@@ -112,28 +112,28 @@ rpc_timeout_upcall_queue(struct work_struct *work)  
int  
rpc_queue_upcall(struct inode *inode, struct rpc_pipe_msg *msg)  
{  
- struct rpc_inode *rpci = RPC_I(inode);  
+ struct rpc_pipe *pipe = RPC_I(inode)->pipe;  
int res = -EPIPE;  
  
- spin_lock(&rpci->pipe->lock);  
- if (rpci->pipe->ops == NULL)  
+ spin_lock(&pipe->lock);  
+ if (pipe->ops == NULL)  
goto out;  
- if (rpci->pipe->nreaders) {  
- list_add_tail(&msg->list, &rpci->pipe->pipe);  
- rpci->pipe->pipelen += msg->len;  
+ if (pipe->nreaders) {  
+ list_add_tail(&msg->list, &pipe->pipe);  
+ pipe->pipelen += msg->len;  
res = 0;  
- } else if (rpci->pipe->flags & RPC_PIPE_WAIT_FOR_OPEN) {  
- if (list_empty(&rpci->pipe->pipe))  
+ } else if (pipe->flags & RPC_PIPE_WAIT_FOR_OPEN) {  
+ if (list_empty(&pipe->pipe))  
queue_delayed_work(rpciod_workqueue,  
- &rpci->pipe->queue_timeout,  
+ &pipe->queue_timeout,  
RPC_UPCALL_TIMEOUT);  
- list_add_tail(&msg->list, &rpci->pipe->pipe);  
- rpci->pipe->pipelen += msg->len;  
+ list_add_tail(&msg->list, &pipe->pipe);
```

```

+ pipe->pipelen += msg->len;
  res = 0;
}
out:
- spin_unlock(&rpci->pipe->lock);
- wake_up(&rpci->pipe->waitq);
+ spin_unlock(&pipe->lock);
+ wake_up(&pipe->waitq);
  return res;
}
EXPORT_SYMBOL_GPL(rpc_queue_upcall);
@@ -201,23 +201,23 @@ rpc_destroy_inode(struct inode *inode)
static int
rpc_pipe_open(struct inode *inode, struct file *filp)
{
- struct rpc_inode *rpci = RPC_I(inode);
+ struct rpc_pipe *pipe = RPC_I(inode)->pipe;
  int first_open;
  int res = -ENXIO;

  mutex_lock(&inode->i_mutex);
- if (rpci->pipe->ops == NULL)
+ if (pipe->ops == NULL)
  goto out;
- first_open = rpci->pipe->nreaders == 0 && rpci->pipe->nwriters == 0;
- if (first_open && rpci->pipe->ops->open_pipe) {
- res = rpci->pipe->ops->open_pipe(inode);
+ first_open = pipe->nreaders == 0 && pipe->nwriters == 0;
+ if (first_open && pipe->ops->open_pipe) {
+ res = pipe->ops->open_pipe(inode);
  if (res)
    goto out;
}
if (filp->f_mode & FMODE_READ)
- rpci->pipe->nreaders++;
+ pipe->nreaders++;
if (filp->f_mode & FMODE_WRITE)
- rpci->pipe->nwriters++;
+ pipe->nwriters++;
  res = 0;
out:
  mutex_unlock(&inode->i_mutex);
@@ -268,39 +268,39 @@ 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_inode *rpci = RPC_I(inode);
+ struct rpc_pipe *pipe = RPC_I(inode)->pipe;

```

```

struct rpc_pipe_msg *msg;
int res = 0;

mutex_lock(&inode->i_mutex);
- if (rpci->pipe->ops == NULL) {
+ if (pipe->ops == NULL) {
    res = -EPIPE;
    goto out_unlock;
}
msg = filp->private_data;
if (msg == NULL) {
- spin_lock(&rpci->pipe->lock);
- if (!list_empty(&rpci->pipe->pipe)) {
- msg = list_entry(rpci->pipe->pipe.next,
+ spin_lock(&pipe->lock);
+ if (!list_empty(&pipe->pipe)) {
+ msg = list_entry(pipe->pipe.next,
    struct rpc_pipe_msg,
    list);
- list_move(&msg->list, &rpci->pipe->in_upcall);
- rpci->pipe->pipelen -= msg->len;
+ list_move(&msg->list, &pipe->in_upcall);
+ pipe->pipelen -= msg->len;
    filp->private_data = msg;
    msg->copied = 0;
}
- spin_unlock(&rpci->pipe->lock);
+ spin_unlock(&pipe->lock);
    if (msg == NULL)
        goto out_unlock;
}
/* NOTE: it is up to the callback to update msg->copied */
- res = rpci->pipe->ops->upcall(filp, msg, buf, len);
+ res = pipe->ops->upcall(filp, msg, buf, len);
    if (res < 0 || msg->len == msg->copied) {
        filp->private_data = NULL;
- spin_lock(&rpci->pipe->lock);
+ spin_lock(&pipe->lock);
        list_del_init(&msg->list);
- spin_unlock(&rpci->pipe->lock);
- rpci->pipe->ops->destroy_msg(msg);
+ spin_unlock(&pipe->lock);
+ pipe->ops->destroy_msg(msg);
    }
out_unlock:
    mutex_unlock(&inode->i_mutex);
@@ -311,13 +311,13 @@ 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_inode *rpci = RPC_I(inode);
+ struct rpc_pipe *pipe = RPC_I(inode)->pipe;
    int res;

    mutex_lock(&inode->i_mutex);
    res = -EPIPE;
- if (rpci->pipe->ops != NULL)
- res = rpci->pipe->ops->downcall(filp, buf, len);
+ if (pipe->ops != NULL)
+ res = pipe->ops->downcall(filp, buf, len);
    mutex_unlock(&inode->i_mutex);
    return res;
}
@@ -325,16 +325,15 @@ 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_inode *rpci;
+ struct rpc_pipe *pipe = RPC_I(filp->f_path.dentry->d_inode)->pipe;
    unsigned int mask = 0;

- rpci = RPC_I(filp->f_path.dentry->d_inode);
- poll_wait(filp, &rpci->pipe->waitq, wait);
+ poll_wait(filp, &pipe->waitq, wait);

    mask = POLLOUT | POLLWRNORM;
- if (rpci->pipe->ops == NULL)
+ if (pipe->ops == NULL)
    mask |= POLLERR | POLLHUP;
- if (filp->private_data || !list_empty(&rpci->pipe->pipe))
+ if (filp->private_data || !list_empty(&pipe->pipe))
    mask |= POLLIN | POLLRDNORM;
    return mask;
}
@@ -343,23 +342,23 @@ 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_inode *rpci = RPC_I(inode);
+ struct rpc_pipe *pipe = RPC_I(inode)->pipe;
    int len;

    switch (cmd) {
    case FIONREAD:
- spin_lock(&rpci->pipe->lock);

```

```

- if (rpci->pipe->ops == NULL) {
- spin_unlock(&rpci->pipe->lock);
+ spin_lock(&pipe->lock);
+ if (pipe->ops == NULL) {
+ spin_unlock(&pipe->lock);
  return -EPIPE;
}
- len = rpci->pipe->pipelen;
+ len = pipe->pipelen;
  if (filp->private_data) {
    struct rpc_pipe_msg *msg;
    msg = filp->private_data;
    len += msg->len - msg->copied;
  }
- spin_unlock(&rpci->pipe->lock);
+ spin_unlock(&pipe->lock);
  return put_user(len, (int __user *)arg);
default:
  return -EINVAL;
@@ -789,7 +788,7 @@ static int rpc_rmdir_depopulate(struct dentry *dentry,
 * @private: private data to associate with the pipe, for the caller's use
 * @ops: operations defining the behavior of the pipe: upcall, downcall,
 * release_pipe, open_pipe, and destroy_msg.
- * @flags: rpc_inode flags
+ * @flags: rpc_pipe flags
 *
 * Data is made available for userspace to read by calls to
 * rpc_queue_upcall(). The actual reads will result in calls to

```
