Subject: [PATCH 2/6] SUNRPC: split SUNPRC PipeFS pipe data and inode creation
Posted by Stanislav Kinsbursky on Tue, 22 Nov 2011 14:45:46 GMT
View Forum Message <> Reply to Message

Generally, pipe data is used only for pipes, and thus allocating space for it
on every RPC inode allocation is redundant. This patch splits private SUNRPC
PipeFS pipe data and inode, makes pipe data allocated only for pipe inodes.
This patch is also is a next step towards to to removing PipeFS inode
references from kernel code other than PipeFS itself.

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

```
---
 include/linux/sunrpc/rpc_pipe_fs.h |   10 +-
 net/sunrpc/auth_gss/auth_gss.c     |   46 ++++----
 net/sunrpc/rpc_pipe.c              |  208 ++++++++++++++++++++----------------
 3 files changed, 142 insertions(+), 122 deletions(-)

diff --git a/include/linux/sunrpc/rpc_pipe_fs.h b/include/linux/sunrpc/rpc_pipe_fs.h
index 8c51471..c2fa330 100644
--- a/include/linux/sunrpc/rpc_pipe_fs.h
+++ b/include/linux/sunrpc/rpc_pipe_fs.h
@@ -21,9 +21,7 @@ struct rpc_pipe_ops {
  void (*destroy_msg)(struct rpc_pipe_msg *);
 };

-struct rpc_inode {
- struct inode vfs_inode;
- void *private;
+struct rpc_pipe {
  struct list_head pipe;
  struct list_head in_upcall;
  struct list_head in_downcall;
@@ -38,6 +36,12 @@ struct rpc_inode {
  spinlock_t lock;
 };

+struct rpc_inode {
+ struct inode vfs_inode;
+ void *private;
+ struct rpc_pipe *pipe;
+};
+
 static inline struct rpc_inode *
 RPC_I(struct inode *inode)
 {
diff --git a/net/sunrpc/auth_gss/auth_gss.c b/net/sunrpc/auth_gss/auth_gss.c
```

```
index 6ba2784..70a7953 100644
--- a/net/sunrpc/auth_gss/auth_gss.c
+++ b/net/sunrpc/auth_gss/auth_gss.c
@@ -112,7 +112,7 @@ gss_put_ctx(struct gss_cl_ctx *ctx)
 /* gss_cred_set_ctx:
  * called by gss_upcall_callback and gss_create_upcall in order
  * to set the gss context. The actual exchange of an old context
- * and a new one is protected by the rpci->lock.
+ * and a new one is protected by the rpci->pipe->lock.
  */
 static void
 gss_cred_set_ctx(struct rpc_cred *cred, struct gss_cl_ctx *ctx)
@@ -297,7 +297,7 @@ static struct gss_upcall_msg *
 __gss_find_upcall(struct rpc_inode *rpci, uid_t uid)
 {
  struct gss_upcall_msg *pos;
- list_for_each_entry(pos, &rpci->in_downcall, list) {
+ list_for_each_entry(pos, &rpci->pipe->in_downcall, list) {
   if (pos->uid != uid)
    continue;
   atomic_inc(&pos->count);
@@ -318,14 +318,14 @@ gss_add_msg(struct gss_upcall_msg *gss_msg)
  struct rpc_inode *rpci = gss_msg->inode;
  struct gss_upcall_msg *old;

- spin_lock(&rpci->lock);
+ spin_lock(&rpci->pipe->lock);
  old = __gss_find_upcall(rpci, gss_msg->uid);
  if (old == NULL) {
   atomic_inc(&gss_msg->count);
-  list_add(&gss_msg->list, &rpci->in_downcall);
+  list_add(&gss_msg->list, &rpci->pipe->in_downcall);
  } else
   gss_msg = old;
- spin_unlock(&rpci->lock);
+ spin_unlock(&rpci->pipe->lock);
  return gss_msg;
 }

@@ -345,10 +345,10 @@ gss_unhash_msg(struct gss_upcall_msg *gss_msg)

  if (list_empty(&gss_msg->list))
   return;
- spin_lock(&rpci->lock);
+ spin_lock(&rpci->pipe->lock);
  if (!list_empty(&gss_msg->list))
   __gss_unhash_msg(gss_msg);
- spin_unlock(&rpci->lock);
```

```
+ spin_unlock(&rpci->pipe->lock);
 }

 static void
@@ -377,9 +377,9 @@ gss_upcall_callback(struct rpc_task *task)
  struct gss_upcall_msg *gss_msg = gss_cred->gc_upcall;
  struct rpc_inode *rpci = gss_msg->inode;

- spin_lock(&rpci->lock);
+ spin_lock(&rpci->pipe->lock);
  gss_handle_downcall_result(gss_cred, gss_msg);
- spin_unlock(&rpci->lock);
+ spin_unlock(&rpci->pipe->lock);
  task->tk_status = gss_msg->msg.errno;
  gss_release_msg(gss_msg);
 }
@@ -526,7 +526,7 @@ gss_refresh_upcall(struct rpc_task *task)
   goto out;
  }
  rpci = gss_msg->inode;
- spin_lock(&rpci->lock);
+ spin_lock(&rpci->pipe->lock);
  if (gss_cred->gc_upcall != NULL)
   rpc_sleep_on(&gss_cred->gc_upcall->rpc_waitqueue, task, NULL);
  else if (gss_msg->ctx == NULL && gss_msg->msg.errno >= 0) {
@@ -539,7 +539,7 @@ gss_refresh_upcall(struct rpc_task *task)
   gss_handle_downcall_result(gss_cred, gss_msg);
   err = gss_msg->msg.errno;
  }
- spin_unlock(&rpci->lock);
+ spin_unlock(&rpci->pipe->lock);
  gss_release_msg(gss_msg);
 out:
  dprintk("RPC: %5u gss_refresh_upcall for uid %u result %d\n",
@@ -577,11 +577,11 @@ retry:
  rpci = gss_msg->inode;
  for (;;) {
   prepare_to_wait(&gss_msg->waitqueue, &wait, TASK_KILLABLE);
-  spin_lock(&rpci->lock);
+  spin_lock(&rpci->pipe->lock);
   if (gss_msg->ctx != NULL || gss_msg->msg.errno < 0) {
    break;
   }
-  spin_unlock(&rpci->lock);
+  spin_unlock(&rpci->pipe->lock);
   if (fatal_signal_pending(current)) {
    err = -ERESTARTSYS;
    goto out_intr;
```

```
@@ -592,7 +592,7 @@ retry:
   gss_cred_set_ctx(cred, gss_msg->ctx);
  else
   err = gss_msg->msg.errno;
- spin_unlock(&rpci->lock);
+ spin_unlock(&rpci->pipe->lock);
 out_intr:
  finish_wait(&gss_msg->waitqueue, &wait);
  gss_release_msg(gss_msg);
@@ -660,14 +660,14 @@ gss_pipe_downcall(struct file *filp, const char __user *src, size_t mlen)

  err = -ENOENT;
  /* Find a matching upcall */
- spin_lock(&rpci->lock);
+ spin_lock(&rpci->pipe->lock);
  gss_msg = __gss_find_upcall(rpci, uid);
  if (gss_msg == NULL) {
-  spin_unlock(&rpci->lock);
+  spin_unlock(&rpci->pipe->lock);
   goto err_put_ctx;
  }
  list_del_init(&gss_msg->list);
- spin_unlock(&rpci->lock);
+ spin_unlock(&rpci->pipe->lock);

  p = gss_fill_context(p, end, ctx, gss_msg->auth->mech);
  if (IS_ERR(p)) {
@@ -695,9 +695,9 @@ gss_pipe_downcall(struct file *filp, const char __user *src, size_t mlen)
  err = mlen;

 err_release_msg:
- spin_lock(&rpci->lock);
+ spin_lock(&rpci->pipe->lock);
  __gss_unhash_msg(gss_msg);
- spin_unlock(&rpci->lock);
+ spin_unlock(&rpci->pipe->lock);
  gss_release_msg(gss_msg);
 err_put_ctx:
  gss_put_ctx(ctx);
@@ -747,19 +747,19 @@ gss_pipe_release(struct inode *inode)
  struct gss_upcall_msg *gss_msg;

 restart:
- spin_lock(&rpci->lock);
- list_for_each_entry(gss_msg, &rpci->in_downcall, list) {
+ spin_lock(&rpci->pipe->lock);
+ list_for_each_entry(gss_msg, &rpci->pipe->in_downcall, list) {
```

```
   if (!list_empty(&gss_msg->msg.list))
    continue;
   gss_msg->msg.errno = -EPIPE;
   atomic_inc(&gss_msg->count);
   __gss_unhash_msg(gss_msg);
-  spin_unlock(&rpci->lock);
+  spin_unlock(&rpci->pipe->lock);
   gss_release_msg(gss_msg);
   goto restart;
  }
- spin_unlock(&rpci->lock);
+ spin_unlock(&rpci->pipe->lock);

  put_pipe_version();
 }
diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index d0ffdf4..a95ba18 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -61,7 +61,7 @@ void rpc_pipefs_notifier_unregister(struct notifier_block *nb)
 }
 EXPORT_SYMBOL_GPL(rpc_pipefs_notifier_unregister);

-static void rpc_purge_list(struct rpc_inode *rpci, struct list_head *head,
+static void rpc_purge_list(struct rpc_pipe *pipe, struct list_head *head,
   void (*destroy_msg)(struct rpc_pipe_msg *), int err)
 {
  struct rpc_pipe_msg *msg;
@@ -74,29 +74,29 @@ static void rpc_purge_list(struct rpc_inode *rpci, struct list_head *head,
   msg->errno = err;
   destroy_msg(msg);
  } while (!list_empty(head));
- wake_up(&rpci->waitq);
+ wake_up(&pipe->waitq);
 }

 static void
 rpc_timeout_upcall_queue(struct work_struct *work)
 {
  LIST_HEAD(free_list);
- struct rpc_inode *rpci =
-  container_of(work, struct rpc_inode, queue_timeout.work);
+ struct rpc_pipe *pipe =
+  container_of(work, struct rpc_pipe, queue_timeout.work);
  void (*destroy_msg)(struct rpc_pipe_msg *);

- spin_lock(&rpci->lock);
- if (rpci->ops == NULL) {
```

```
-  spin_unlock(&rpci->lock);
+ spin_lock(&pipe->lock);
+ if (pipe->ops == NULL) {
+  spin_unlock(&pipe->lock);
   return;
  }
- destroy_msg = rpci->ops->destroy_msg;
- if (rpci->nreaders == 0) {
-  list_splice_init(&rpci->pipe, &free_list);
-  rpci->pipelen = 0;
+ destroy_msg = pipe->ops->destroy_msg;
+ if (pipe->nreaders == 0) {
+  list_splice_init(&pipe->pipe, &free_list);
+  pipe->pipelen = 0;
  }
- spin_unlock(&rpci->lock);
- rpc_purge_list(rpci, &free_list, destroy_msg, -ETIMEDOUT);
+ spin_unlock(&pipe->lock);
+ rpc_purge_list(pipe, &free_list, destroy_msg, -ETIMEDOUT);
 }

 /**
@@ -115,25 +115,25 @@ rpc_queue_upcall(struct inode *inode, struct rpc_pipe_msg *msg)
  struct rpc_inode *rpci = RPC_I(inode);
  int res = -EPIPE;

- spin_lock(&rpci->lock);
- if (rpci->ops == NULL)
+ spin_lock(&rpci->pipe->lock);
+ if (rpci->pipe->ops == NULL)
   goto out;
- if (rpci->nreaders) {
-  list_add_tail(&msg->list, &rpci->pipe);
-  rpci->pipelen += msg->len;
+ if (rpci->pipe->nreaders) {
+  list_add_tail(&msg->list, &rpci->pipe->pipe);
+  rpci->pipe->pipelen += msg->len;
   res = 0;
- } else if (rpci->flags & RPC_PIPE_WAIT_FOR_OPEN) {
-  if (list_empty(&rpci->pipe))
+ } else if (rpci->pipe->flags & RPC_PIPE_WAIT_FOR_OPEN) {
+  if (list_empty(&rpci->pipe->pipe))
    queue_delayed_work(rpciod_workqueue,
-     &rpci->queue_timeout,
+     &rpci->pipe->queue_timeout,
      RPC_UPCALL_TIMEOUT);
-  list_add_tail(&msg->list, &rpci->pipe);
-  rpci->pipelen += msg->len;
```

```
+  list_add_tail(&msg->list, &rpci->pipe->pipe);
+  rpci->pipe->pipelen += msg->len;
   res = 0;
  }
 out:
- spin_unlock(&rpci->lock);
- wake_up(&rpci->waitq);
+ spin_unlock(&rpci->pipe->lock);
+ wake_up(&rpci->pipe->waitq);
  return res;
 }
 EXPORT_SYMBOL_GPL(rpc_queue_upcall);
@@ -147,27 +147,27 @@ rpc_inode_setowner(struct inode *inode, void *private)
 static void
 rpc_close_pipes(struct inode *inode)
 {
- struct rpc_inode *rpci = RPC_I(inode);
+ struct rpc_pipe *pipe = RPC_I(inode)->pipe;
  const struct rpc_pipe_ops *ops;
  int need_release;

  mutex_lock(&inode->i_mutex);
- ops = rpci->ops;
+ ops = pipe->ops;
  if (ops != NULL) {
   LIST_HEAD(free_list);
-  spin_lock(&rpci->lock);
-  need_release = rpci->nreaders != 0 || rpci->nwriters != 0;
-  rpci->nreaders = 0;
-  list_splice_init(&rpci->in_upcall, &free_list);
-  list_splice_init(&rpci->pipe, &free_list);
-  rpci->pipelen = 0;
-  rpci->ops = NULL;
-  spin_unlock(&rpci->lock);
-  rpc_purge_list(rpci, &free_list, ops->destroy_msg, -EPIPE);
-  rpci->nwriters = 0;
+  spin_lock(&pipe->lock);
+  need_release = pipe->nreaders != 0 || pipe->nwriters != 0;
+  pipe->nreaders = 0;
+  list_splice_init(&pipe->in_upcall, &free_list);
+  list_splice_init(&pipe->pipe, &free_list);
+  pipe->pipelen = 0;
+  pipe->ops = NULL;
+  spin_unlock(&pipe->lock);
+  rpc_purge_list(pipe, &free_list, ops->destroy_msg, -EPIPE);
+  pipe->nwriters = 0;
   if (need_release && ops->release_pipe)
    ops->release_pipe(inode);
```

```
-  cancel_delayed_work_sync(&rpci->queue_timeout);
+  cancel_delayed_work_sync(&pipe->queue_timeout);
   }
   rpc_inode_setowner(inode, NULL);
   mutex_unlock(&inode->i_mutex);
@@ -188,6 +188,7 @@ rpc_i_callback(struct rcu_head *head)
 {
   struct inode *inode = container_of(head, struct inode, i_rcu);
   INIT_LIST_HEAD(&inode->i_dentry);
+ kfree(RPC_I(inode)->pipe);
   kmem_cache_free(rpc_inode_cachep, RPC_I(inode));
 }

@@ -205,18 +206,18 @@ rpc_pipe_open(struct inode *inode, struct file *filp)
   int res = -ENXIO;

   mutex_lock(&inode->i_mutex);
- if (rpci->ops == NULL)
+ if (rpci->pipe->ops == NULL)
    goto out;
- first_open = rpci->nreaders == 0 && rpci->nwriters == 0;
- if (first_open && rpci->ops->open_pipe) {
-  res = rpci->ops->open_pipe(inode);
+ 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);
    if (res)
     goto out;
   }
   if (filp->f_mode & FMODE_READ)
-  rpci->nreaders++;
+  rpci->pipe->nreaders++;
   if (filp->f_mode & FMODE_WRITE)
-  rpci->nwriters++;
+  rpci->pipe->nwriters++;
   res = 0;
 out:
   mutex_unlock(&inode->i_mutex);
@@ -226,38 +227,38 @@ out:
 static int
 rpc_pipe_release(struct inode *inode, struct file *filp)
 {
- struct rpc_inode *rpci = RPC_I(inode);
+ struct rpc_pipe *pipe = RPC_I(inode)->pipe;
   struct rpc_pipe_msg *msg;
   int last_close;

   mutex_lock(&inode->i_mutex);
```

```
- if (rpci->ops == NULL)
+ if (pipe->ops == NULL)
    goto out;
   msg = filp->private_data;
   if (msg != NULL) {
-   spin_lock(&rpci->lock);
+   spin_lock(&pipe->lock);
    msg->errno = -EAGAIN;
    list_del_init(&msg->list);
-   spin_unlock(&rpci->lock);
-   rpci->ops->destroy_msg(msg);
+   spin_unlock(&pipe->lock);
+   pipe->ops->destroy_msg(msg);
   }
   if (filp->f_mode & FMODE_WRITE)
-   rpci->nwriters --;
+   pipe->nwriters --;
   if (filp->f_mode & FMODE_READ) {
-   rpci->nreaders --;
-   if (rpci->nreaders == 0) {
+   pipe->nreaders --;
+   if (pipe->nreaders == 0) {
     LIST_HEAD(free_list);
-    spin_lock(&rpci->lock);
-    list_splice_init(&rpci->pipe, &free_list);
-    rpci->pipelen = 0;
-    spin_unlock(&rpci->lock);
-    rpc_purge_list(rpci, &free_list,
-      rpci->ops->destroy_msg, -EAGAIN);
+    spin_lock(&pipe->lock);
+    list_splice_init(&pipe->pipe, &free_list);
+    pipe->pipelen = 0;
+    spin_unlock(&pipe->lock);
+    rpc_purge_list(pipe, &free_list,
+      pipe->ops->destroy_msg, -EAGAIN);
    }
   }
- last_close = rpci->nwriters == 0 && rpci->nreaders == 0;
- if (last_close && rpci->ops->release_pipe)
-   rpci->ops->release_pipe(inode);
+ last_close = pipe->nwriters == 0 && pipe->nreaders == 0;
+ if (last_close && pipe->ops->release_pipe)
+   pipe->ops->release_pipe(inode);
 out:
   mutex_unlock(&inode->i_mutex);
   return 0;
@@ -272,34 +273,34 @@ rpc_pipe_read(struct file *filp, char __user *buf, size_t len, loff_t
*offset)
```

```
  int res = 0;

  mutex_lock(&inode->i_mutex);
- if (rpci->ops == NULL) {
+ if (rpci->pipe->ops == NULL) {
   res = -EPIPE;
   goto out_unlock;
  }
  msg = filp->private_data;
  if (msg == NULL) {
-  spin_lock(&rpci->lock);
-  if (!list_empty(&rpci->pipe)) {
-   msg = list_entry(rpci->pipe.next,
+  spin_lock(&rpci->pipe->lock);
+  if (!list_empty(&rpci->pipe->pipe)) {
+   msg = list_entry(rpci->pipe->pipe.next,
       struct rpc_pipe_msg,
       list);
-   list_move(&msg->list, &rpci->in_upcall);
-   rpci->pipelen -= msg->len;
+   list_move(&msg->list, &rpci->pipe->in_upcall);
+   rpci->pipe->pipelen -= msg->len;
    filp->private_data = msg;
    msg->copied = 0;
   }
-  spin_unlock(&rpci->lock);
+  spin_unlock(&rpci->pipe->lock);
   if (msg == NULL)
    goto out_unlock;
  }
  /* NOTE: it is up to the callback to update msg->copied */
- res = rpci->ops->upcall(filp, msg, buf, len);
+ res = rpci->pipe->ops->upcall(filp, msg, buf, len);
  if (res < 0 || msg->len == msg->copied) {
   filp->private_data = NULL;
-  spin_lock(&rpci->lock);
+  spin_lock(&rpci->pipe->lock);
   list_del_init(&msg->list);
-  spin_unlock(&rpci->lock);
-  rpci->ops->destroy_msg(msg);
+  spin_unlock(&rpci->pipe->lock);
+  rpci->pipe->ops->destroy_msg(msg);
  }
 out_unlock:
  mutex_unlock(&inode->i_mutex);
@@ -315,8 +316,8 @@ rpc_pipe_write(struct file *filp, const char __user *buf, size_t len, loff_t *of

  mutex_lock(&inode->i_mutex);
```

```
  res = -EPIPE;
- if (rpci->ops != NULL)
-  res = rpci->ops->downcall(filp, buf, len);
+ if (rpci->pipe->ops != NULL)
+  res = rpci->pipe->ops->downcall(filp, buf, len);
  mutex_unlock(&inode->i_mutex);
  return res;
 }
@@ -328,12 +329,12 @@ rpc_pipe_poll(struct file *filp, struct poll_table_struct *wait)
  unsigned int mask = 0;

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

  mask = POLLOUT | POLLWRNORM;
- if (rpci->ops == NULL)
+ if (rpci->pipe->ops == NULL)
   mask |= POLLERR | POLLHUP;
- if (filp->private_data || !list_empty(&rpci->pipe))
+ if (filp->private_data || !list_empty(&rpci->pipe->pipe))
   mask |= POLLIN | POLLRDNORM;
  return mask;
 }
@@ -347,18 +348,18 @@ rpc_pipe_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)

  switch (cmd) {
  case FIONREAD:
-  spin_lock(&rpci->lock);
-  if (rpci->ops == NULL) {
-   spin_unlock(&rpci->lock);
+  spin_lock(&rpci->pipe->lock);
+  if (rpci->pipe->ops == NULL) {
+   spin_unlock(&rpci->pipe->lock);
    return -EPIPE;
   }
-  len = rpci->pipelen;
+  len = rpci->pipe->pipelen;
   if (filp->private_data) {
    struct rpc_pipe_msg *msg;
    msg = filp->private_data;
    len += msg->len - msg->copied;
   }
-  spin_unlock(&rpci->lock);
+  spin_unlock(&rpci->pipe->lock);
   return put_user(len, (int __user *)arg);
  default:
   return -EINVAL;
```

```
@@ -543,6 +544,23 @@ static int __rpc_mkdir(struct inode *dir, struct dentry *dentry,
  return 0;
 }

+static void
+init_pipe(struct rpc_pipe *pipe)
+{
+ pipe->nreaders = 0;
+ pipe->nwriters = 0;
+ INIT_LIST_HEAD(&pipe->in_upcall);
+ 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;
+ spin_lock_init(&pipe->lock);
+
+}
+
 static int __rpc_mkpipe(struct inode *dir, struct dentry *dentry,
    umode_t mode,
    const struct file_operations *i_fop,
@@ -550,16 +568,24 @@ static int __rpc_mkpipe(struct inode *dir, struct dentry *dentry,
    const struct rpc_pipe_ops *ops,
    int flags)
 {
+ struct rpc_pipe *pipe;
  struct rpc_inode *rpci;
  int err;

+ pipe = kzalloc(sizeof(struct rpc_pipe), GFP_KERNEL);
+ if (!pipe)
+  return -ENOMEM;
+ init_pipe(pipe);
  err = __rpc_create_common(dir, dentry, S_IFIFO | mode, i_fop, private);
- if (err)
+ if (err) {
+  kfree(pipe);
   return err;
+ }
  rpci = RPC_I(dentry->d_inode);
  rpci->private = private;
- rpci->flags = flags;
- rpci->ops = ops;
+ rpci->pipe = pipe;
+ rpci->pipe->flags = flags;
```

```
+ rpci->pipe->ops = ops;
  fsnotify_create(dir, dentry);
  return 0;
 }
@@ -1123,17 +1149,7 @@ init_once(void *foo)

  inode_init_once(&rpci->vfs_inode);
  rpci->private = NULL;
- rpci->nreaders = 0;
- rpci->nwriters = 0;
- INIT_LIST_HEAD(&rpci->in_upcall);
- INIT_LIST_HEAD(&rpci->in_downcall);
- INIT_LIST_HEAD(&rpci->pipe);
- rpci->pipelen = 0;
- init_waitqueue_head(&rpci->waitq);
- INIT_DELAYED_WORK(&rpci->queue_timeout,
-       rpc_timeout_upcall_queue);
- rpci->ops = NULL;
- spin_lock_init(&rpci->lock);
+ rpci->pipe = NULL;
 }

 int register_rpc_pipefs(void)
```