Subject: [PATCH v2 0/6] SUNPRC: cleanup PipeFS for network-namespace-aware users
Posted by Stanislav Kinsbursky on Mon, 26 Dec 2011 11:44:22 GMT
View Forum Message <> Reply to Message

This patch-set was created in context of clone of git
branch: git://git.linux-nfs.org/projects/trondmy/nfs-2.6.git.
tag: v3.2-rc7

v2:
1) rebased on tag "v3.2-rc7"
2) some cleanup and minor changes

This is cleanup precursor patch set. It's required for easier further
implementation of network-namespace-aware SUNRPC PipeFS pipes creators.
Generally, this patch set split SUNRPC PipeFS logic into two parts: working
with pipes (new rpc_pipe structure with link to PipeFS dentry) and
creating/destroying PipeFS dentries (old rpc_inode structure with link to
rpc_pipe).
With this patch-set kernel PipeFS pipes users initially creates
rpc_pipe data and then creates Pipefs dentries. Later these dentries will be
created in notifier callbacks on PipeFS mount event from user-space and in
network namespace operations for such modules like nfs and blocklayout.

The following series consists of:

Stanislav Kinsbursky (6):
      SUNRPC: replace inode lock with pipe lock for RPC PipeFS operations
      SUNRPC: split SUNPRC PipeFS pipe data and inode creation
      SUNRPC: cleanup PipeFS redundant RPC inode usage
      SUNPRC: cleanup RPC PipeFS pipes upcall interface
      SUNRPC: cleanup GSS pipes usage
      SUNRPC: split SUNPRC PipeFS dentry and private pipe data creation

```
 fs/nfs/blocklayout/blocklayout.c    |   16 ++
 fs/nfs/blocklayout/blocklayout.h    |    2
 fs/nfs/blocklayout/blocklayoutdev.c |    2
 fs/nfs/blocklayout/blocklayoutdm.c  |    2
 fs/nfs/idmap.c                      |   28 +++-
 include/linux/sunrpc/rpc_pipe_fs.h  |   20 ++-
 net/sunrpc/auth_gss/auth_gss.c      |  130 +++++++++++--------
 net/sunrpc/rpc_pipe.c               |  258 +++++++++++++++++++----------------
 8 files changed, 262 insertions(+), 196 deletions(-)
```

--

Signature

Subject: [PATCH v2 4/6] SUNPRC: cleanup RPC PipeFS pipes upcall interface
Posted by Stanislav Kinsbursky on Mon, 26 Dec 2011 11:45:23 GMT
View Forum Message <> Reply to Message

RPC pipe upcall doesn't requires only private pipe data. Thus RPC inode
references in this code can be removed.

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

---
 fs/nfs/blocklayout/blocklayoutdev.c | 2 +-
 fs/nfs/blocklayout/blocklayoutdm.c | 2 +-
 fs/nfs/idmap.c | 4 ++--
 include/linux/sunrpc/rpc_pipe_fs.h | 2 +-
 net/sunrpc/auth_gss/auth_gss.c | 3 +--
 net/sunrpc/rpc_pipe.c | 3 +--
 6 files changed, 7 insertions(+), 9 deletions(-)

```
diff --git a/fs/nfs/blocklayout/blocklayoutdev.c b/fs/nfs/blocklayout/blocklayoutdev.c
index d08ba91..8101919 100644
--- a/fs/nfs/blocklayout/blocklayoutdev.c
+++ b/fs/nfs/blocklayout/blocklayoutdev.c
@@ -146,7 +146,7 @@ nfs4_blk_decode_device(struct nfs_server *server,

  dprintk("%s CALLING USERSPACE DAEMON\n", __func__);
  add_wait_queue(&bl_wq, &wq);
- rc = rpc_queue_upcall(bl_device_pipe->d_inode, &msg);
+ rc = rpc_queue_upcall(RPC_I(bl_device_pipe->d_inode)->pipe, &msg);
  if (rc < 0) {
   remove_wait_queue(&bl_wq, &wq);
   rv = ERR_PTR(rc);
diff --git a/fs/nfs/blocklayout/blocklayoutdm.c b/fs/nfs/blocklayout/blocklayoutdm.c
index d055c75..3c38244 100644
--- a/fs/nfs/blocklayout/blocklayoutdm.c
+++ b/fs/nfs/blocklayout/blocklayoutdm.c
@@ -66,7 +66,7 @@ static void dev_remove(dev_t dev)
  msg.len = sizeof(bl_msg) + bl_msg.totallen;

  add_wait_queue(&bl_wq, &wq);
- if (rpc_queue_upcall(bl_device_pipe->d_inode, &msg) < 0) {
+ if (rpc_queue_upcall(RPC_I(bl_device_pipe->d_inode)->pipe, &msg) < 0) {
   remove_wait_queue(&bl_wq, &wq);
   goto out;
  }
diff --git a/fs/nfs/idmap.c b/fs/nfs/idmap.c
```

```
index 47d1c6f..4194129 100644
--- a/fs/nfs/idmap.c
+++ b/fs/nfs/idmap.c
@@ -506,7 +506,7 @@ nfs_idmap_id(struct idmap *idmap, struct idmap_hashtable *h,
  msg.len = sizeof(*im);

  add_wait_queue(&idmap->idmap_wq, &wq);
- if (rpc_queue_upcall(idmap->idmap_dentry->d_inode, &msg) < 0) {
+ if (rpc_queue_upcall(RPC_I(idmap->idmap_dentry->d_inode)->pipe, &msg) < 0) {
   remove_wait_queue(&idmap->idmap_wq, &wq);
   goto out;
  }
@@ -567,7 +567,7 @@ nfs_idmap_name(struct idmap *idmap, struct idmap_hashtable *h,

  add_wait_queue(&idmap->idmap_wq, &wq);

- if (rpc_queue_upcall(idmap->idmap_dentry->d_inode, &msg) < 0) {
+ if (rpc_queue_upcall(RPC_I(idmap->idmap_dentry->d_inode)->pipe, &msg) < 0) {
   remove_wait_queue(&idmap->idmap_wq, &wq);
   goto out;
  }
diff --git a/include/linux/sunrpc/rpc_pipe_fs.h b/include/linux/sunrpc/rpc_pipe_fs.h
index c2fa330..ad78bea 100644
--- a/include/linux/sunrpc/rpc_pipe_fs.h
+++ b/include/linux/sunrpc/rpc_pipe_fs.h
@@ -64,7 +64,7 @@ extern void rpc_put_sb_net(const struct net *net);

 extern ssize_t rpc_pipe_generic_upcall(struct file *, struct rpc_pipe_msg *,
           char __user *, size_t);
-extern int rpc_queue_upcall(struct inode *, struct rpc_pipe_msg *);
+extern int rpc_queue_upcall(struct rpc_pipe *, struct rpc_pipe_msg *);

 struct rpc_clnt;
 extern struct dentry *rpc_create_client_dir(struct dentry *, struct qstr *, struct rpc_clnt *);
diff --git a/net/sunrpc/auth_gss/auth_gss.c b/net/sunrpc/auth_gss/auth_gss.c
index 89fc914..5ba678d 100644
--- a/net/sunrpc/auth_gss/auth_gss.c
+++ b/net/sunrpc/auth_gss/auth_gss.c
@@ -475,8 +475,7 @@ gss_setup_upcall(struct rpc_clnt *clnt, struct gss_auth *gss_auth, struct
rpc_cr
   return gss_new;
  gss_msg = gss_add_msg(gss_new);
  if (gss_msg == gss_new) {
-  struct inode *inode = &gss_new->inode->vfs_inode;
-  int res = rpc_queue_upcall(inode, &gss_new->msg);
+  int res = rpc_queue_upcall(gss_new->inode->pipe, &gss_new->msg);
   if (res) {
    gss_unhash_msg(gss_new);
```

```
    gss_msg = ERR_PTR(res);
diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index cecdc1f..dc24af3 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -130,9 +130,8 @@ EXPORT_SYMBOL_GPL(rpc_pipe_generic_upcall);
  * initialize the fields of @msg (other than @msg->list) appropriately.
  */
 int
-rpc_queue_upcall(struct inode *inode, struct rpc_pipe_msg *msg)
+rpc_queue_upcall(struct rpc_pipe *pipe, struct rpc_pipe_msg *msg)
 {
- struct rpc_pipe *pipe = RPC_I(inode)->pipe;
  int res = -EPIPE;

  spin_lock(&pipe->lock);
```

Subject: [PATCH v2 3/6] SUNRPC: cleanup PipeFS redundant RPC inode usage
Posted by Stanislav Kinsbursky on Mon, 26 Dec 2011 11:45:25 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 d4a1f86..cecdc1f 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -132,28 +132,28 @@ EXPORT_SYMBOL_GPL(rpc_pipe_generic_upcall);
 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);
@@ -221,23 +221,23 @@ 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);
@@ -288,39 +288,39 @@ 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);
@@ -331,13 +331,13 @@ 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;
 }
@@ -345,16 +345,15 @@ 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;
  }
@@ -363,23 +362,23 @@ 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;
@@ -809,7 +808,7 @@ 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
```

Subject: [PATCH v2 1/6] SUNRPC: replace inode lock with pipe lock for RPC PipeFS operations
Posted by Stanislav Kinsbursky on Mon, 26 Dec 2011 11:45:26 GMT
View Forum Message <> Reply to Message

Currenly, inode i_lock is used to provide concurrent access to SUNPRC PipeFS pipes. It looks redundant, since now other use of inode is present in most of these places and thus can be easely replaced, which will allow to remove most of inode references from PipeFS code. This is a first step towards 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 | 1 +
 net/sunrpc/auth_gss/auth_gss.c | 57 ++++++++++++++++++++----------------
 net/sunrpc/rpc_pipe.c | 38 ++++++++++++++-----------
 3 files changed, 48 insertions(+), 48 deletions(-)

```
diff --git a/include/linux/sunrpc/rpc_pipe_fs.h b/include/linux/sunrpc/rpc_pipe_fs.h
index f32490c..8c51471 100644
--- a/include/linux/sunrpc/rpc_pipe_fs.h
+++ b/include/linux/sunrpc/rpc_pipe_fs.h
@@ -35,6 +35,7 @@ struct rpc_inode {
  int flags;
  struct delayed_work queue_timeout;
  const struct rpc_pipe_ops *ops;
+ spinlock_t lock;
 };

 static inline struct rpc_inode *
diff --git a/net/sunrpc/auth_gss/auth_gss.c b/net/sunrpc/auth_gss/auth_gss.c
index afb5655..0eb36e6 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 inode->i_lock.
+ * and a new one is protected by the rpci->lock.
  */
 static void
 gss_cred_set_ctx(struct rpc_cred *cred, struct gss_cl_ctx *ctx)
@@ -316,17 +316,16 @@ static inline struct gss_upcall_msg *
 gss_add_msg(struct gss_upcall_msg *gss_msg)
 {
  struct rpc_inode *rpci = gss_msg->inode;
- struct inode *inode = &rpci->vfs_inode;
```

```
  struct gss_upcall_msg *old;

- spin_lock(&inode->i_lock);
+ spin_lock(&rpci->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);
  } else
   gss_msg = old;
- spin_unlock(&inode->i_lock);
+ spin_unlock(&rpci->lock);
  return gss_msg;
 }

@@ -342,14 +341,14 @@ __gss_unhash_msg(struct gss_upcall_msg *gss_msg)
 static void
 gss_unhash_msg(struct gss_upcall_msg *gss_msg)
 {
- struct inode *inode = &gss_msg->inode->vfs_inode;
+ struct rpc_inode *rpci = gss_msg->inode;

  if (list_empty(&gss_msg->list))
   return;
- spin_lock(&inode->i_lock);
+ spin_lock(&rpci->lock);
  if (!list_empty(&gss_msg->list))
   __gss_unhash_msg(gss_msg);
- spin_unlock(&inode->i_lock);
+ spin_unlock(&rpci->lock);
 }

 static void
@@ -376,11 +375,11 @@ gss_upcall_callback(struct rpc_task *task)
  struct gss_cred *gss_cred = container_of(task->tk_rqstp->rq_cred,
    struct gss_cred, gc_base);
  struct gss_upcall_msg *gss_msg = gss_cred->gc_upcall;
- struct inode *inode = &gss_msg->inode->vfs_inode;
+ struct rpc_inode *rpci = gss_msg->inode;

- spin_lock(&inode->i_lock);
+ spin_lock(&rpci->lock);
  gss_handle_downcall_result(gss_cred, gss_msg);
- spin_unlock(&inode->i_lock);
+ spin_unlock(&rpci->lock);
  task->tk_status = gss_msg->msg.errno;
  gss_release_msg(gss_msg);
 }
```

```
@@ -508,7 +507,7 @@ gss_refresh_upcall(struct rpc_task *task)
  struct gss_cred *gss_cred = container_of(cred,
    struct gss_cred, gc_base);
  struct gss_upcall_msg *gss_msg;
- struct inode *inode;
+ struct rpc_inode *rpci;
  int err = 0;

  dprintk("RPC: %5u gss_refresh_upcall for uid %u\n", task->tk_pid,
@@ -526,8 +525,8 @@ gss_refresh_upcall(struct rpc_task *task)
   err = PTR_ERR(gss_msg);
   goto out;
  }
- inode = &gss_msg->inode->vfs_inode;
- spin_lock(&inode->i_lock);
+ rpci = gss_msg->inode;
+ spin_lock(&rpci->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) {
@@ -540,7 +539,7 @@ gss_refresh_upcall(struct rpc_task *task)
   gss_handle_downcall_result(gss_cred, gss_msg);
   err = gss_msg->msg.errno;
  }
- spin_unlock(&inode->i_lock);
+ spin_unlock(&rpci->lock);
  gss_release_msg(gss_msg);
 out:
  dprintk("RPC: %5u gss_refresh_upcall for uid %u result %d\n",
@@ -551,7 +550,7 @@ out:
 gss_create_upcall(struct gss_auth *gss_auth, struct gss_cred *gss_cred)
 {
- struct inode *inode;
+ struct rpc_inode *rpci;
  struct rpc_cred *cred = &gss_cred->gc_base;
  struct gss_upcall_msg *gss_msg;
  DEFINE_WAIT(wait);
@@ -575,14 +574,14 @@ retry:
   err = PTR_ERR(gss_msg);
   goto out;
  }
- inode = &gss_msg->inode->vfs_inode;
+ rpci = gss_msg->inode;
  for (;;) {
   prepare_to_wait(&gss_msg->waitqueue, &wait, TASK_KILLABLE);
-  spin_lock(&inode->i_lock);
+  spin_lock(&rpci->lock);
```

```
   if (gss_msg->ctx != NULL || gss_msg->msg.errno < 0) {
    break;
   }
-  spin_unlock(&inode->i_lock);
+  spin_unlock(&rpci->lock);
   if (fatal_signal_pending(current)) {
    err = -ERESTARTSYS;
    goto out_intr;
@@ -593,7 +592,7 @@ retry:
   gss_cred_set_ctx(cred, gss_msg->ctx);
  else
   err = gss_msg->msg.errno;
- spin_unlock(&inode->i_lock);
+ spin_unlock(&rpci->lock);
 out_intr:
  finish_wait(&gss_msg->waitqueue, &wait);
  gss_release_msg(gss_msg);
@@ -611,7 +610,7 @@ gss_pipe_downcall(struct file *filp, const char __user *src, size_t mlen)
  const void *p, *end;
  void *buf;
  struct gss_upcall_msg *gss_msg;
- struct inode *inode = filp->f_path.dentry->d_inode;
+ struct rpc_inode *rpci = RPC_I(filp->f_dentry->d_inode);
  struct gss_cl_ctx *ctx;
  uid_t uid;
  ssize_t err = -EFBIG;
@@ -641,14 +640,14 @@ gss_pipe_downcall(struct file *filp, const char __user *src, size_t mlen)

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

  p = gss_fill_context(p, end, ctx, gss_msg->auth->mech);
  if (IS_ERR(p)) {
@@ -676,9 +675,9 @@ gss_pipe_downcall(struct file *filp, const char __user *src, size_t mlen)
  err = mlen;
```

```
 err_release_msg:
- spin_lock(&inode->i_lock);
+ spin_lock(&rpci->lock);
  __gss_unhash_msg(gss_msg);
- spin_unlock(&inode->i_lock);
+ spin_unlock(&rpci->lock);
  gss_release_msg(gss_msg);
 err_put_ctx:
  gss_put_ctx(ctx);
@@ -728,7 +727,7 @@ gss_pipe_release(struct inode *inode)
  struct gss_upcall_msg *gss_msg;

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

   if (!list_empty(&gss_msg->msg.list))
@@ -736,11 +735,11 @@ restart:
   gss_msg->msg.errno = -EPIPE;
   atomic_inc(&gss_msg->count);
   __gss_unhash_msg(gss_msg);
-  spin_unlock(&inode->i_lock);
+  spin_unlock(&rpci->lock);
   gss_release_msg(gss_msg);
   goto restart;
  }
- spin_unlock(&inode->i_lock);
+ spin_unlock(&rpci->lock);

  put_pipe_version();
 }
diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index 0cafd59..0e4d75c 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -83,12 +83,11 @@ 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 inode *inode = &rpci->vfs_inode;
  void (*destroy_msg)(struct rpc_pipe_msg *);

- spin_lock(&inode->i_lock);
+ spin_lock(&rpci->lock);
  if (rpci->ops == NULL) {
-  spin_unlock(&inode->i_lock);
+  spin_unlock(&rpci->lock);
```

```
   return;
  }
  destroy_msg = rpci->ops->destroy_msg;
@@ -96,7 +95,7 @@ rpc_timeout_upcall_queue(struct work_struct *work)
   list_splice_init(&rpci->pipe, &free_list);
   rpci->pipelen = 0;
  }
- spin_unlock(&inode->i_lock);
+ spin_unlock(&rpci->lock);
  rpc_purge_list(rpci, &free_list, destroy_msg, -ETIMEDOUT);
 }

@@ -136,7 +135,7 @@ rpc_queue_upcall(struct inode *inode, struct rpc_pipe_msg *msg)
  struct rpc_inode *rpci = RPC_I(inode);
  int res = -EPIPE;

- spin_lock(&inode->i_lock);
+ spin_lock(&rpci->lock);
  if (rpci->ops == NULL)
   goto out;
  if (rpci->nreaders) {
@@ -153,7 +152,7 @@ rpc_queue_upcall(struct inode *inode, struct rpc_pipe_msg *msg)
   res = 0;
  }
 out:
- spin_unlock(&inode->i_lock);
+ spin_unlock(&rpci->lock);
  wake_up(&rpci->waitq);
  return res;
 }
@@ -176,14 +175,14 @@ rpc_close_pipes(struct inode *inode)
  ops = rpci->ops;
  if (ops != NULL) {
   LIST_HEAD(free_list);
-  spin_lock(&inode->i_lock);
+  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(&inode->i_lock);
+  spin_unlock(&rpci->lock);
   rpc_purge_list(rpci, &free_list, ops->destroy_msg, -EPIPE);
   rpci->nwriters = 0;
   if (need_release && ops->release_pipe)
@@ -256,10 +255,10 @@ rpc_pipe_release(struct inode *inode, struct file *filp)
```

```
   goto out;
  msg = filp->private_data;
  if (msg != NULL) {
-  spin_lock(&inode->i_lock);
+  spin_lock(&rpci->lock);
   msg->errno = -EAGAIN;
   list_del_init(&msg->list);
-  spin_unlock(&inode->i_lock);
+  spin_unlock(&rpci->lock);
   rpci->ops->destroy_msg(msg);
  }
  if (filp->f_mode & FMODE_WRITE)
@@ -268,10 +267,10 @@ rpc_pipe_release(struct inode *inode, struct file *filp)
   rpci->nreaders --;
   if (rpci->nreaders == 0) {
    LIST_HEAD(free_list);
-   spin_lock(&inode->i_lock);
+   spin_lock(&rpci->lock);
    list_splice_init(&rpci->pipe, &free_list);
    rpci->pipelen = 0;
-   spin_unlock(&inode->i_lock);
+   spin_unlock(&rpci->lock);
    rpc_purge_list(rpci, &free_list,
      rpci->ops->destroy_msg, -EAGAIN);
   }
@@ -299,7 +298,7 @@ rpc_pipe_read(struct file *filp, char __user *buf, size_t len, loff_t *offset)
  }
  msg = filp->private_data;
  if (msg == NULL) {
-  spin_lock(&inode->i_lock);
+  spin_lock(&rpci->lock);
   if (!list_empty(&rpci->pipe)) {
    msg = list_entry(rpci->pipe.next,
      struct rpc_pipe_msg,
@@ -309,7 +308,7 @@ rpc_pipe_read(struct file *filp, char __user *buf, size_t len, loff_t *offset)
    filp->private_data = msg;
    msg->copied = 0;
   }
-  spin_unlock(&inode->i_lock);
+  spin_unlock(&rpci->lock);
   if (msg == NULL)
    goto out_unlock;
  }
@@ -317,9 +316,9 @@ rpc_pipe_read(struct file *filp, char __user *buf, size_t len, loff_t *offset)
  res = rpci->ops->upcall(filp, msg, buf, len);
  if (res < 0 || msg->len == msg->copied) {
   filp->private_data = NULL;
-  spin_lock(&inode->i_lock);
```

```
+  spin_lock(&rpci->lock);
   list_del_init(&msg->list);
-  spin_unlock(&inode->i_lock);
+  spin_unlock(&rpci->lock);
   rpci->ops->destroy_msg(msg);
  }
 out_unlock:
@@ -368,9 +367,9 @@ rpc_pipe_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)

  switch (cmd) {
  case FIONREAD:
-  spin_lock(&inode->i_lock);
+  spin_lock(&rpci->lock);
   if (rpci->ops == NULL) {
-   spin_unlock(&inode->i_lock);
+   spin_unlock(&rpci->lock);
    return -EPIPE;
   }
   len = rpci->pipelen;
@@ -379,7 +378,7 @@ rpc_pipe_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
    msg = filp->private_data;
    len += msg->len - msg->copied;
   }
-  spin_unlock(&inode->i_lock);
+  spin_unlock(&rpci->lock);
   return put_user(len, (int __user *)arg);
  default:
   return -EINVAL;
@@ -1154,6 +1153,7 @@ init_once(void *foo)
  INIT_DELAYED_WORK(&rpci->queue_timeout,
       rpc_timeout_upcall_queue);
  rpci->ops = NULL;
+ spin_lock_init(&rpci->lock);
 }

 int register_rpc_pipefs(void)
```

Subject: [PATCH v2 2/6] SUNRPC: split SUNPRC PipeFS pipe data and inode creation

Posted by Stanislav Kinsbursky on Mon, 26 Dec 2011 11:45:28 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 0eb36e6..89fc914 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);
@@ -640,14 +640,14 @@ 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)) {
@@ -675,9 +675,9 @@ 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);
@@ -727,19 +727,19 @@ 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 0e4d75c..d4a1f86 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);
 }

 ssize_t rpc_pipe_generic_upcall(struct file *filp, struct rpc_pipe_msg *msg,
@@ -135,25 +135,25 @@ 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);
@@ -167,27 +167,27 @@ 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);
@@ -208,6 +208,7 @@ 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));
 }

@@ -225,18 +226,18 @@ 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);
@@ -246,38 +247,38 @@ out:
 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;
@@ -292,34 +293,34 @@ 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);
@@ -335,8 +336,8 @@ 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;
 }
@@ -348,12 +349,12 @@ 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;
 }
@@ -367,18 +368,18 @@ 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;
@@ -563,6 +564,23 @@ 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,
@@ -570,16 +588,24 @@ 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;
 }
@@ -1143,17 +1169,7 @@ 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)
```

Subject: [PATCH v2 5/6] SUNRPC: cleanup GSS pipes usage
Posted by Stanislav Kinsbursky on Mon, 26 Dec 2011 11:45:28 GMT
View Forum Message <> Reply to Message

Currently gss auth holds RPC inode pointer which is now redundant since it
requires only pipes operations which takes private pipe data as an argument.
Thus this code can be cleaned and all references to RPC inode can be replaced
with privtae pipe data references.

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

```
---
 net/sunrpc/auth_gss/auth_gss.c |   76 ++++++++++++++++++++++++++----------------
 1 files changed, 38 insertions(+), 38 deletions(-)

diff --git a/net/sunrpc/auth_gss/auth_gss.c b/net/sunrpc/auth_gss/auth_gss.c
index 5ba678d..f99fde5 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->pipe->lock.
+ * and a new one is protected by the pipe->lock.
  */
 static void
 gss_cred_set_ctx(struct rpc_cred *cred, struct gss_cl_ctx *ctx)
@@ -251,7 +251,7 @@ struct gss_upcall_msg {
  struct rpc_pipe_msg msg;
  struct list_head list;
  struct gss_auth *auth;
- struct rpc_inode *inode;
```

```
+ struct rpc_pipe *pipe;
  struct rpc_wait_queue rpc_waitqueue;
  wait_queue_head_t waitqueue;
  struct gss_cl_ctx *ctx;
@@ -294,10 +294,10 @@ gss_release_msg(struct gss_upcall_msg *gss_msg)
 }

 static struct gss_upcall_msg *
-__gss_find_upcall(struct rpc_inode *rpci, uid_t uid)
+__gss_find_upcall(struct rpc_pipe *pipe, uid_t uid)
 {
  struct gss_upcall_msg *pos;
- list_for_each_entry(pos, &rpci->pipe->in_downcall, list) {
+ list_for_each_entry(pos, &pipe->in_downcall, list) {
   if (pos->uid != uid)
    continue;
   atomic_inc(&pos->count);
@@ -315,17 +315,17 @@ __gss_find_upcall(struct rpc_inode *rpci, uid_t uid)
 static inline struct gss_upcall_msg *
 gss_add_msg(struct gss_upcall_msg *gss_msg)
 {
- struct rpc_inode *rpci = gss_msg->inode;
+ struct rpc_pipe *pipe = gss_msg->pipe;
  struct gss_upcall_msg *old;

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

@@ -341,14 +341,14 @@ __gss_unhash_msg(struct gss_upcall_msg *gss_msg)
 static void
 gss_unhash_msg(struct gss_upcall_msg *gss_msg)
 {
- struct rpc_inode *rpci = gss_msg->inode;
+ struct rpc_pipe *pipe = gss_msg->pipe;

  if (list_empty(&gss_msg->list))
```

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

 static void
@@ -375,11 +375,11 @@ gss_upcall_callback(struct rpc_task *task)
  struct gss_cred *gss_cred = container_of(task->tk_rqstp->rq_cred,
    struct gss_cred, gc_base);
  struct gss_upcall_msg *gss_msg = gss_cred->gc_upcall;
- struct rpc_inode *rpci = gss_msg->inode;
+ struct rpc_pipe *pipe = gss_msg->pipe;

- spin_lock(&rpci->pipe->lock);
+ spin_lock(&pipe->lock);
  gss_handle_downcall_result(gss_cred, gss_msg);
- spin_unlock(&rpci->pipe->lock);
+ spin_unlock(&pipe->lock);
  task->tk_status = gss_msg->msg.errno;
  gss_release_msg(gss_msg);
 }
@@ -451,7 +451,7 @@ gss_alloc_msg(struct gss_auth *gss_auth, uid_t uid, struct rpc_clnt *clnt,
   kfree(gss_msg);
   return ERR_PTR(vers);
  }
- gss_msg->inode = RPC_I(gss_auth->dentry[vers]->d_inode);
+ gss_msg->pipe = RPC_I(gss_auth->dentry[vers]->d_inode)->pipe;
  INIT_LIST_HEAD(&gss_msg->list);
  rpc_init_wait_queue(&gss_msg->rpc_waitqueue, "RPCSEC_GSS upcall waitq");
  init_waitqueue_head(&gss_msg->waitqueue);
@@ -475,7 +475,7 @@ gss_setup_upcall(struct rpc_clnt *clnt, struct gss_auth *gss_auth, struct
rpc_cr
   return gss_new;
  gss_msg = gss_add_msg(gss_new);
  if (gss_msg == gss_new) {
-  int res = rpc_queue_upcall(gss_new->inode->pipe, &gss_new->msg);
+  int res = rpc_queue_upcall(gss_new->pipe, &gss_new->msg);
   if (res) {
    gss_unhash_msg(gss_new);
    gss_msg = ERR_PTR(res);
@@ -506,7 +506,7 @@ gss_refresh_upcall(struct rpc_task *task)
  struct gss_cred *gss_cred = container_of(cred,
    struct gss_cred, gc_base);
  struct gss_upcall_msg *gss_msg;
```

```
- struct rpc_inode *rpci;
+ struct rpc_pipe *pipe;
  int err = 0;

  dprintk("RPC: %5u gss_refresh_upcall for uid %u\n", task->tk_pid,
@@ -524,8 +524,8 @@ gss_refresh_upcall(struct rpc_task *task)
   err = PTR_ERR(gss_msg);
   goto out;
  }
- rpci = gss_msg->inode;
- spin_lock(&rpci->pipe->lock);
+ pipe = gss_msg->pipe;
+ spin_lock(&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) {
@@ -538,7 +538,7 @@ gss_refresh_upcall(struct rpc_task *task)
   gss_handle_downcall_result(gss_cred, gss_msg);
   err = gss_msg->msg.errno;
  }
- spin_unlock(&rpci->pipe->lock);
+ spin_unlock(&pipe->lock);
  gss_release_msg(gss_msg);
 out:
  dprintk("RPC: %5u gss_refresh_upcall for uid %u result %d\n",
@@ -549,7 +549,7 @@ out:
 static inline int
 gss_create_upcall(struct gss_auth *gss_auth, struct gss_cred *gss_cred)
 {
- struct rpc_inode *rpci;
+ struct rpc_pipe *pipe;
  struct rpc_cred *cred = &gss_cred->gc_base;
  struct gss_upcall_msg *gss_msg;
  DEFINE_WAIT(wait);
@@ -573,14 +573,14 @@ retry:
   err = PTR_ERR(gss_msg);
   goto out;
  }
- rpci = gss_msg->inode;
+ pipe = gss_msg->pipe;
  for (;;) {
   prepare_to_wait(&gss_msg->waitqueue, &wait, TASK_KILLABLE);
-  spin_lock(&rpci->pipe->lock);
+  spin_lock(&pipe->lock);
   if (gss_msg->ctx != NULL || gss_msg->msg.errno < 0) {
    break;
   }
-  spin_unlock(&rpci->pipe->lock);
```

```
+  spin_unlock(&pipe->lock);
   if (fatal_signal_pending(current)) {
    err = -ERESTARTSYS;
    goto out_intr;
@@ -591,7 +591,7 @@ retry:
   gss_cred_set_ctx(cred, gss_msg->ctx);
  else
   err = gss_msg->msg.errno;
- spin_unlock(&rpci->pipe->lock);
+ spin_unlock(&pipe->lock);
 out_intr:
  finish_wait(&gss_msg->waitqueue, &wait);
  gss_release_msg(gss_msg);
@@ -609,7 +609,7 @@ gss_pipe_downcall(struct file *filp, const char __user *src, size_t mlen)
  const void *p, *end;
  void *buf;
  struct gss_upcall_msg *gss_msg;
- struct rpc_inode *rpci = RPC_I(filp->f_dentry->d_inode);
+ struct rpc_pipe *pipe = RPC_I(filp->f_dentry->d_inode)->pipe;
  struct gss_cl_ctx *ctx;
  uid_t uid;
  ssize_t err = -EFBIG;
@@ -639,14 +639,14 @@ gss_pipe_downcall(struct file *filp, const char __user *src, size_t mlen)

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

  p = gss_fill_context(p, end, ctx, gss_msg->auth->mech);
  if (IS_ERR(p)) {
@@ -674,9 +674,9 @@ gss_pipe_downcall(struct file *filp, const char __user *src, size_t mlen)
  err = mlen;

 err_release_msg:
- spin_lock(&rpci->pipe->lock);
+ spin_lock(&pipe->lock);
  __gss_unhash_msg(gss_msg);
```

```
- spin_unlock(&rpci->pipe->lock);
+ spin_unlock(&pipe->lock);
  gss_release_msg(gss_msg);
 err_put_ctx:
  gss_put_ctx(ctx);
@@ -722,23 +722,23 @@ static int gss_pipe_open_v1(struct inode *inode)
 static void
 gss_pipe_release(struct inode *inode)
 {
- struct rpc_inode *rpci = RPC_I(inode);
+ struct rpc_pipe *pipe = RPC_I(inode)->pipe;
  struct gss_upcall_msg *gss_msg;

 restart:
- spin_lock(&rpci->pipe->lock);
- list_for_each_entry(gss_msg, &rpci->pipe->in_downcall, list) {
+ spin_lock(&pipe->lock);
+ list_for_each_entry(gss_msg, &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->pipe->lock);
+  spin_unlock(&pipe->lock);
   gss_release_msg(gss_msg);
   goto restart;
  }
- spin_unlock(&rpci->pipe->lock);
+ spin_unlock(&pipe->lock);

  put_pipe_version();
 }
```

Subject: [PATCH v2 6/6] SUNRPC: split SUNPRC PipeFS dentry and private pipe data creation

Posted by Stanislav Kinsbursky on Mon, 26 Dec 2011 11:46:30 GMT

View Forum Message <> Reply to Message

This patch is a final step towards to removing PipeFS inode references from kernel code other than PipeFS itself. It makes all kernel SUNRPC PipeFS users depends on pipe private data, which state depend on their specific operations, etc.
This patch completes SUNRPC PipeFS preparations and allows to create pipe private data and PipeFS dentries independently.
Next step will be making SUNPRC PipeFS dentries allocated by SUNRPC PipeFS

network namespace aware routines.

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

---
 fs/nfs/blocklayout/blocklayout.c | 16 ++++++++--
 fs/nfs/blocklayout/blocklayout.h | 2 +
 fs/nfs/blocklayout/blocklayoutdev.c | 2 +
 fs/nfs/blocklayout/blocklayoutdm.c | 2 +
 fs/nfs/idmap.c | 28 +++++++++++++++++-----
 include/linux/sunrpc/rpc_pipe_fs.h | 7 +++--
 net/sunrpc/auth_gss/auth_gss.c | 54 ++++++++++++++++++++++++++++++------------
 net/sunrpc/rpc_pipe.c | 54 ++++++++++++++++++++++++++++++------------
 8 files changed, 107 insertions(+), 58 deletions(-)

diff --git a/fs/nfs/blocklayout/blocklayout.c b/fs/nfs/blocklayout/blocklayout.c
index 281ae95..2f4ede1 100644
--- a/fs/nfs/blocklayout/blocklayout.c
+++ b/fs/nfs/blocklayout/blocklayout.c
@@ -46,7 +46,7 @@ MODULE_LICENSE("GPL");
 MODULE_AUTHOR("Andy Adamson <andros@citi.umich.edu>");
 MODULE_DESCRIPTION("The NFSv4.1 pNFS Block layout driver");

-struct dentry *bl_device_pipe;
+struct rpc_pipe *bl_device_pipe;
 wait_queue_head_t bl_wq;

 static void print_page(struct page *page)
@@ -991,16 +991,23 @@ static int __init nfs4blocklayout_init(void)
  if (ret)
   goto out_putrpc;

- bl_device_pipe = rpc_mkpipe(path.dentry, "blocklayout", NULL,
-        &bl_upcall_ops, 0);
+ bl_device_pipe = rpc_mkpipe_data(&bl_upcall_ops, 0);
  path_put(&path);
  if (IS_ERR(bl_device_pipe)) {
   ret = PTR_ERR(bl_device_pipe);
   goto out_putrpc;
  }
+ bl_device_pipe->dentry = rpc_mkpipe_dentry(path.dentry, "blocklayout",
+   NULL, bl_device_pipe);
+ if (IS_ERR(bl_device_pipe->dentry)) {
+ ret = PTR_ERR(bl_device_pipe->dentry);
+ goto out_destroy_pipe;
+ }
 out:
  return ret;

```
+out_destroy_pipe:
+ rpc_destroy_pipe_data(bl_device_pipe);
 out_putrpc:
  rpc_put_mount();
 out_remove:
@@ -1014,7 +1021,8 @@ static void __exit nfs4blocklayout_exit(void)
         __func__);

  pnfs_unregister_layoutdriver(&blocklayout_type);
- rpc_unlink(bl_device_pipe);
+ rpc_unlink(bl_device_pipe->dentry);
+ rpc_destroy_pipe_data(bl_device_pipe);
  rpc_put_mount();
 }

diff --git a/fs/nfs/blocklayout/blocklayout.h b/fs/nfs/blocklayout/blocklayout.h
index 42acf7e..046b513 100644
--- a/fs/nfs/blocklayout/blocklayout.h
+++ b/fs/nfs/blocklayout/blocklayout.h
@@ -159,7 +159,7 @@ struct bl_msg_hdr {
  u16 totallen; /* length of entire message, including hdr itself */
 };

-extern struct dentry *bl_device_pipe;
+extern struct rpc_pipe *bl_device_pipe;
 extern wait_queue_head_t bl_wq;

 #define BL_DEVICE_UMOUNT               0x0 /* Umount--delete devices */
diff --git a/fs/nfs/blocklayout/blocklayoutdev.c b/fs/nfs/blocklayout/blocklayoutdev.c
index 8101919..949b624 100644
--- a/fs/nfs/blocklayout/blocklayoutdev.c
+++ b/fs/nfs/blocklayout/blocklayoutdev.c
@@ -146,7 +146,7 @@ nfs4_blk_decode_device(struct nfs_server *server,

  dprintk("%s CALLING USERSPACE DAEMON\n", __func__);
  add_wait_queue(&bl_wq, &wq);
- rc = rpc_queue_upcall(RPC_I(bl_device_pipe->d_inode)->pipe, &msg);
+ rc = rpc_queue_upcall(bl_device_pipe, &msg);
  if (rc < 0) {
   remove_wait_queue(&bl_wq, &wq);
   rv = ERR_PTR(rc);
diff --git a/fs/nfs/blocklayout/blocklayoutdm.c b/fs/nfs/blocklayout/blocklayoutdm.c
index 3c38244..631f254 100644
--- a/fs/nfs/blocklayout/blocklayoutdm.c
+++ b/fs/nfs/blocklayout/blocklayoutdm.c
@@ -66,7 +66,7 @@ static void dev_remove(dev_t dev)
  msg.len = sizeof(bl_msg) + bl_msg.totallen;
```

```
  add_wait_queue(&bl_wq, &wq);
- if (rpc_queue_upcall(RPC_I(bl_device_pipe->d_inode)->pipe, &msg) < 0) {
+ if (rpc_queue_upcall(bl_device_pipe, &msg) < 0) {
    remove_wait_queue(&bl_wq, &wq);
    goto out;
  }
diff --git a/fs/nfs/idmap.c b/fs/nfs/idmap.c
index 4194129..096e374 100644
--- a/fs/nfs/idmap.c
+++ b/fs/nfs/idmap.c
@@ -327,7 +327,7 @@ struct idmap_hashtable {
 };

 struct idmap {
- struct dentry  *idmap_dentry;
+ struct rpc_pipe  *idmap_pipe;
   wait_queue_head_t idmap_wq;
   struct idmap_msg idmap_im;
   struct mutex  idmap_lock; /* Serializes upcalls */
@@ -352,6 +352,7 @@ int
 nfs_idmap_new(struct nfs_client *clp)
 {
   struct idmap *idmap;
+ struct rpc_pipe *pipe;
   int error;

   BUG_ON(clp->cl_idmap != NULL);
@@ -360,14 +361,23 @@ nfs_idmap_new(struct nfs_client *clp)
   if (idmap == NULL)
    return -ENOMEM;

- idmap->idmap_dentry = rpc_mkpipe(clp->cl_rpcclient->cl_path.dentry,
-    "idmap", idmap, &idmap_upcall_ops, 0);
- if (IS_ERR(idmap->idmap_dentry)) {
-  error = PTR_ERR(idmap->idmap_dentry);
+ pipe = rpc_mkpipe_data(&idmap_upcall_ops, 0);
+ if (IS_ERR(pipe)) {
+  error = PTR_ERR(pipe);
    kfree(idmap);
    return error;
   }

+ if (clp->cl_rpcclient->cl_path.dentry)
+  pipe->dentry = rpc_mkpipe_dentry(clp->cl_rpcclient->cl_path.dentry,
+    "idmap", idmap, pipe);
+ if (IS_ERR(pipe->dentry)) {
+  error = PTR_ERR(pipe->dentry);
```

```
+  rpc_destroy_pipe_data(pipe);
+  kfree(idmap);
+  return error;
+ }
+ idmap->idmap_pipe = pipe;
  mutex_init(&idmap->idmap_lock);
  mutex_init(&idmap->idmap_im_lock);
  init_waitqueue_head(&idmap->idmap_wq);
@@ -385,7 +395,9 @@ nfs_idmap_delete(struct nfs_client *clp)

  if (!idmap)
   return;
- rpc_unlink(idmap->idmap_dentry);
+ if (idmap->idmap_pipe->dentry)
+  rpc_unlink(idmap->idmap_pipe->dentry);
+ rpc_destroy_pipe_data(idmap->idmap_pipe);
  clp->cl_idmap = NULL;
  kfree(idmap);
 }
@@ -506,7 +518,7 @@ nfs_idmap_id(struct idmap *idmap, struct idmap_hashtable *h,
  msg.len = sizeof(*im);

  add_wait_queue(&idmap->idmap_wq, &wq);
- if (rpc_queue_upcall(RPC_I(idmap->idmap_dentry->d_inode)->pipe, &msg) < 0) {
+ if (rpc_queue_upcall(idmap->idmap_pipe, &msg) < 0) {
   remove_wait_queue(&idmap->idmap_wq, &wq);
   goto out;
  }
@@ -567,7 +579,7 @@ nfs_idmap_name(struct idmap *idmap, struct idmap_hashtable *h,

  add_wait_queue(&idmap->idmap_wq, &wq);

- if (rpc_queue_upcall(RPC_I(idmap->idmap_dentry->d_inode)->pipe, &msg) < 0) {
+ if (rpc_queue_upcall(idmap->idmap_pipe, &msg) < 0) {
   remove_wait_queue(&idmap->idmap_wq, &wq);
   goto out;
  }
diff --git a/include/linux/sunrpc/rpc_pipe_fs.h b/include/linux/sunrpc/rpc_pipe_fs.h
index ad78bea..0808ed2 100644
--- a/include/linux/sunrpc/rpc_pipe_fs.h
+++ b/include/linux/sunrpc/rpc_pipe_fs.h
@@ -34,6 +34,7 @@ struct rpc_pipe {
  struct delayed_work queue_timeout;
  const struct rpc_pipe_ops *ops;
  spinlock_t lock;
+ struct dentry *dentry;
 };
```

```
 struct rpc_inode {
@@ -77,8 +78,10 @@ extern struct dentry *rpc_create_cache_dir(struct dentry *,
         struct cache_detail *);
 extern void rpc_remove_cache_dir(struct dentry *);

-extern struct dentry *rpc_mkpipe(struct dentry *, const char *, void *,
-      const struct rpc_pipe_ops *, int flags);
+struct rpc_pipe *rpc_mkpipe_data(const struct rpc_pipe_ops *ops, int flags);
+void rpc_destroy_pipe_data(struct rpc_pipe *pipe);
+extern struct dentry *rpc_mkpipe_dentry(struct dentry *, const char *, void *,
+      struct rpc_pipe *);
 extern int rpc_unlink(struct dentry *);
 extern struct vfsmount *rpc_get_mount(void);
 extern void rpc_put_mount(void);
diff --git a/net/sunrpc/auth_gss/auth_gss.c b/net/sunrpc/auth_gss/auth_gss.c
index f99fde5..7544305 100644
--- a/net/sunrpc/auth_gss/auth_gss.c
+++ b/net/sunrpc/auth_gss/auth_gss.c
@@ -81,7 +81,7 @@ struct gss_auth {
    * mechanism (for example, "krb5") and exists for
    * backwards-compatibility with older gssd's.
    */
- struct dentry *dentry[2];
+ struct rpc_pipe *pipe[2];
 };

 /* pipe_version >= 0 if and only if someone has a pipe open. */
@@ -451,7 +451,7 @@ gss_alloc_msg(struct gss_auth *gss_auth, uid_t uid, struct rpc_clnt *clnt,
    kfree(gss_msg);
    return ERR_PTR(vers);
   }
- gss_msg->pipe = RPC_I(gss_auth->dentry[vers]->d_inode)->pipe;
+ gss_msg->pipe = gss_auth->pipe[vers];
   INIT_LIST_HEAD(&gss_msg->list);
   rpc_init_wait_queue(&gss_msg->rpc_waitqueue, "RPCSEC_GSS upcall waitq");
   init_waitqueue_head(&gss_msg->waitqueue);
@@ -801,21 +801,33 @@ gss_create(struct rpc_clnt *clnt, rpc_authflavor_t flavor)
    * that we supported only the old pipe.  So we instead create
    * the new pipe first.
    */
- gss_auth->dentry[1] = rpc_mkpipe(clnt->cl_path.dentry,
-       "gssd",
-       clnt, &gss_upcall_ops_v1,
-       RPC_PIPE_WAIT_FOR_OPEN);
- if (IS_ERR(gss_auth->dentry[1])) {
-  err = PTR_ERR(gss_auth->dentry[1]);
+ gss_auth->pipe[1] = rpc_mkpipe_data(&gss_upcall_ops_v1,
+         RPC_PIPE_WAIT_FOR_OPEN);
```

```
+ if (IS_ERR(gss_auth->pipe[1])) {
+  err = PTR_ERR(gss_auth->pipe[1]);
   goto err_put_mech;
  }

- gss_auth->dentry[0] = rpc_mkpipe(clnt->cl_path.dentry,
-      gss_auth->mech->gm_name,
-      clnt, &gss_upcall_ops_v0,
-      RPC_PIPE_WAIT_FOR_OPEN);
- if (IS_ERR(gss_auth->dentry[0])) {
-  err = PTR_ERR(gss_auth->dentry[0]);
+ gss_auth->pipe[0] = rpc_mkpipe_data(&gss_upcall_ops_v0,
+         RPC_PIPE_WAIT_FOR_OPEN);
+ if (IS_ERR(gss_auth->pipe[0])) {
+  err = PTR_ERR(gss_auth->pipe[0]);
+  goto err_destroy_pipe_1;
+ }
+
+ gss_auth->pipe[1]->dentry = rpc_mkpipe_dentry(clnt->cl_path.dentry,
+            "gssd",
+            clnt, gss_auth->pipe[1]);
+ if (IS_ERR(gss_auth->pipe[1]->dentry)) {
+  err = PTR_ERR(gss_auth->pipe[1]->dentry);
+  goto err_destroy_pipe_0;
+ }
+
+ gss_auth->pipe[0]->dentry = rpc_mkpipe_dentry(clnt->cl_path.dentry,
+            gss_auth->mech->gm_name,
+            clnt, gss_auth->pipe[0]);
+ if (IS_ERR(gss_auth->pipe[0]->dentry)) {
+  err = PTR_ERR(gss_auth->pipe[0]->dentry);
   goto err_unlink_pipe_1;
  }
  err = rpcauth_init_credcache(auth);
@@ -824,9 +836,13 @@ gss_create(struct rpc_clnt *clnt, rpc_authflavor_t flavor)

  return auth;
 err_unlink_pipe_0:
- rpc_unlink(gss_auth->dentry[0]);
+ rpc_unlink(gss_auth->pipe[0]->dentry);
 err_unlink_pipe_1:
- rpc_unlink(gss_auth->dentry[1]);
+ rpc_unlink(gss_auth->pipe[1]->dentry);
+err_destroy_pipe_0:
+ rpc_destroy_pipe_data(gss_auth->pipe[0]);
+err_destroy_pipe_1:
+ rpc_destroy_pipe_data(gss_auth->pipe[1]);
 err_put_mech:
```

```
  gss_mech_put(gss_auth->mech);
 err_free:
@@ -839,8 +855,10 @@ out_dec:
 static void
 gss_free(struct gss_auth *gss_auth)
 {
- rpc_unlink(gss_auth->dentry[1]);
- rpc_unlink(gss_auth->dentry[0]);
+ rpc_unlink(gss_auth->pipe[0]->dentry);
+ rpc_unlink(gss_auth->pipe[1]->dentry);
+ rpc_destroy_pipe_data(gss_auth->pipe[0]);
+ rpc_destroy_pipe_data(gss_auth->pipe[1]);
  gss_mech_put(gss_auth->mech);

  kfree(gss_auth);
diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index dc24af3..d3a0d26 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -207,7 +207,6 @@ 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));
 }

@@ -576,34 +575,44 @@ init_pipe(struct rpc_pipe *pipe)
        rpc_timeout_upcall_queue);
  pipe->ops = NULL;
  spin_lock_init(&pipe->lock);
+ pipe->dentry = NULL;
+}

+void rpc_destroy_pipe_data(struct rpc_pipe *pipe)
+{
+ kfree(pipe);
 }
+EXPORT_SYMBOL_GPL(rpc_destroy_pipe_data);

-static int __rpc_mkpipe(struct inode *dir, struct dentry *dentry,
-   umode_t mode,
-   const struct file_operations *i_fop,
-   void *private,
-   const struct rpc_pipe_ops *ops,
-   int flags)
+struct rpc_pipe *rpc_mkpipe_data(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;
+  return ERR_PTR(-ENOMEM);
  init_pipe(pipe);
+ pipe->ops = ops;
+ pipe->flags = flags;
+ return pipe;
+}
+EXPORT_SYMBOL_GPL(rpc_mkpipe_data);
+
+static int __rpc_mkpipe_dentry(struct inode *dir, struct dentry *dentry,
+         umode_t mode,
+         const struct file_operations *i_fop,
+         void *private,
+         struct rpc_pipe *pipe)
+{
+ struct rpc_inode *rpci;
+ int err;
+
  err = __rpc_create_common(dir, dentry, S_IFIFO | mode, i_fop, private);
- if (err) {
-  kfree(pipe);
+ if (err)
   return err;
- }
  rpci = RPC_I(dentry->d_inode);
  rpci->private = private;
  rpci->pipe = pipe;
- rpci->pipe->flags = flags;
- rpci->pipe->ops = ops;
  fsnotify_create(dir, dentry);
  return 0;
 }
@@ -820,9 +829,8 @@ static int rpc_rmdir_depopulate(struct dentry *dentry,
  * The @private argument passed here will be available to all these methods
  * from the file pointer, via RPC_I(file->f_dentry->d_inode)->private.
  */
-struct dentry *rpc_mkpipe(struct dentry *parent, const char *name,
-     void *private, const struct rpc_pipe_ops *ops,
-     int flags)
+struct dentry *rpc_mkpipe_dentry(struct dentry *parent, const char *name,
+     void *private, struct rpc_pipe *pipe)
 {
```

```
  struct dentry *dentry;
  struct inode *dir = parent->d_inode;
@@ -830,9 +838,9 @@ struct dentry *rpc_mkpipe(struct dentry *parent, const char *name,
  struct qstr q;
  int err;

- if (ops->upcall == NULL)
+ if (pipe->ops->upcall == NULL)
   umode &= ~S_IRUGO;
- if (ops->downcall == NULL)
+ if (pipe->ops->downcall == NULL)
   umode &= ~S_IWUGO;

  q.name = name;
@@ -843,8 +851,8 @@ struct dentry *rpc_mkpipe(struct dentry *parent, const char *name,
  dentry = __rpc_lookup_create_exclusive(parent, &q);
  if (IS_ERR(dentry))
   goto out;
- err = __rpc_mkpipe(dir, dentry, umode, &rpc_pipe_fops,
-       private, ops, flags);
+ err = __rpc_mkpipe_dentry(dir, dentry, umode, &rpc_pipe_fops,
+       private, pipe);
  if (err)
   goto out_err;
 out:
@@ -857,7 +865,7 @@ out_err:
    err);
  goto out;
 }
-EXPORT_SYMBOL_GPL(rpc_mkpipe);
+EXPORT_SYMBOL_GPL(rpc_mkpipe_dentry);

 /**
  * rpc_unlink - remove a pipe
```

Subject: Re: [PATCH v2 0/6] SUNPRC: cleanup PipeFS for network-namespace-aware users

Posted by Stanislav Kinsbursky on Mon, 26 Dec 2011 11:50:35 GMT

View Forum Message <> Reply to Message

Hello, Trond.
This patch set depend on
"SUNRPC: initial part of making pipefs work in net ns"
and
"SUNRPC: remove non-exclusive pipe creation from RPC pipefs"

I'm not sure about did you applied them already or not, so I've rebased and

resent them too.

--
Best regards,
Stanislav Kinsbursky