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:
 static inline int
 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)
```