Subject: [PATCH v2 0/4] SUNRPC: several fixes around PipeFS objects
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 15:50:47 GMT
View Forum Message <> Reply to Message

v2:
1) Prior to calling PipeFS dentry rountines (for both type of clients - SUNPRC
and NFS) get the client and drop the list lock instead of replacing per-net
locks by mutexes.

First two pathes fixes lockdep warnings and next two - dereferencing of
released pipe data on eventfd close and in file operations.

The following series consists of:

---

Stanislav Kinsbursky (4):
    SUNRPC: release per-net clients lock before calling PipeFS dentries creation
    NFS: release per-net clients lock before calling PipeFS dentries creation
    SUNRPC: check RPC inode's pipe reference before dereferencing
    SUNRPC: move waitq from RPC pipe to RPC inode

 fs/nfs/client.c                 |    2 +
 fs/nfs/idmap.c                  |    8 +++-
 include/linux/sunrpc/rpc_pipe_fs.h |    2 +
 net/sunrpc/clnt.c               |   10 ++++-
 net/sunrpc/rpc_pipe.c           |   71 ++++++++++++++++++++++++++----------
 5 files changed, 60 insertions(+), 33 deletions(-)

Subject: [PATCH v2 1/4] SUNRPC: release per-net clients lock before calling PipeFS dentries creation
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 15:50:55 GMT
View Forum Message <> Reply to Message

Lockdep is sad otherwise, because inode mutex is taken on PipeFS dentry
creation, which can be called on mount notification, where this per-net client
lock is taken on clients list walk.

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

---
 net/sunrpc/clnt.c |   10 +++++++---
 1 files changed, 7 insertions(+), 3 deletions(-)

diff --git a/net/sunrpc/clnt.c b/net/sunrpc/clnt.c
index bb7ed2f3..ddb5741 100644

```
--- a/net/sunrpc/clnt.c
+++ b/net/sunrpc/clnt.c
@@ -84,7 +84,7 @@ static void rpc_register_client(struct rpc_clnt *clnt)
  struct sunrpc_net *sn = net_generic(clnt->cl_xprt->xprt_net, sunrpc_net_id);

  spin_lock(&sn->rpc_client_lock);
- list_add(&clnt->cl_clients, &sn->all_clients);
+ list_add_tail(&clnt->cl_clients, &sn->all_clients);
  spin_unlock(&sn->rpc_client_lock);
 }

@@ -208,15 +208,19 @@ static int rpc_pipefs_event(struct notifier_block *nb, unsigned long
event,
        void *ptr)
 {
  struct super_block *sb = ptr;
- struct rpc_clnt *clnt;
+ struct rpc_clnt *clnt, *tmp;
  int error = 0;
  struct sunrpc_net *sn = net_generic(sb->s_fs_info, sunrpc_net_id);

  spin_lock(&sn->rpc_client_lock);
- list_for_each_entry(clnt, &sn->all_clients, cl_clients) {
+ list_for_each_entry_safe(clnt, tmp, &sn->all_clients, cl_clients) {
+  atomic_inc(&clnt->cl_count);
+  spin_unlock(&sn->rpc_client_lock);
   error = __rpc_pipefs_event(clnt, event, sb);
+  rpc_release_client(clnt);
   if (error)
    break;
+  spin_lock(&sn->rpc_client_lock);
  }
  spin_unlock(&sn->rpc_client_lock);
  return error;
```

---

## Subject: [PATCH v2 2/4] NFS: release per-net clients lock before calling PipeFS dentries creation

Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 15:51:04 GMT

View Forum Message <> Reply to Message

Lockdep is sad otherwise, because inode mutex is taken on PipeFS dentry creation, which can be called on mount notification, where this per-net client lock is taken on clients list walk.

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

---

```
 fs/nfs/client.c |    2 +-
 fs/nfs/idmap.c  |    8 ++++++--
 2 files changed, 7 insertions(+), 3 deletions(-)

diff --git a/fs/nfs/client.c b/fs/nfs/client.c
index 8563585..6aeb6b3 100644
--- a/fs/nfs/client.c
+++ b/fs/nfs/client.c
@@ -538,7 +538,7 @@ nfs_get_client(const struct nfs_client_initdata *cl_init,
  /* install a new client and return with it unready */
 install_client:
  clp = new;
- list_add(&clp->cl_share_link, &nn->nfs_client_list);
+ list_add_tail(&clp->cl_share_link, &nn->nfs_client_list);
  spin_unlock(&nn->nfs_client_lock);

  error = cl_init->rpc_ops->init_client(clp, timeparms, ip_addr,
diff --git a/fs/nfs/idmap.c b/fs/nfs/idmap.c
index b5c6d8e..8a9e7a4 100644
--- a/fs/nfs/idmap.c
+++ b/fs/nfs/idmap.c
@@ -558,16 +558,20 @@ static int rpc_pipefs_event(struct notifier_block *nb, unsigned long
event,
 {
  struct super_block *sb = ptr;
  struct nfs_net *nn = net_generic(sb->s_fs_info, nfs_net_id);
- struct nfs_client *clp;
+ struct nfs_client *clp, *tmp;
  int error = 0;

  spin_lock(&nn->nfs_client_lock);
- list_for_each_entry(clp, &nn->nfs_client_list, cl_share_link) {
+ list_for_each_entry_safe(clp, tmp, &nn->nfs_client_list, cl_share_link) {
   if (clp->rpc_ops != &nfs_v4_clientops)
    continue;
+  atomic_inc(&clp->cl_count);
+  spin_unlock(&nn->nfs_client_lock);
   error = __rpc_pipefs_event(clp, event, sb);
+  nfs_put_client(clp);
   if (error)
    break;
+  spin_lock(&nn->nfs_client_lock);
  }
  spin_unlock(&nn->nfs_client_lock);
  return error;
```

Subject: [PATCH v2 3/4] SUNRPC: check RPC inode's pipe reference before dereferencing
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 15:51:12 GMT
View Forum Message <> Reply to Message

There are 2 tightly bound objects: pipe data (created for kernel needs, has
reference to dentry, which depends on PipeFS mount/umount) and PipeFS
dentry/inode pair (created on mount for user-space needs). They both
independently may have or have not a valid reference to each other.
This means, that we have to make sure, that pipe->dentry reference is valid on
upcalls, and dentry->pipe reference is valid on downcalls. The latter check is
absent - my fault.
IOW, PipeFS dentry can be opened by some process (rpc.idmapd for example), but
it's pipe data can belong to NFS mount, which was unmounted already and thus
pipe data was destroyed.
To fix this, pipe reference have to be set to NULL on rpc_unlink() and checked
on PipeFS file operations instead of pipe->dentry check.

Note: PipeFS "poll" file operation will be updated in next patch, because it's
logic is more complicated.

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

---
 net/sunrpc/rpc_pipe.c |   32 ++++++++++++++++++++++----------
 1 files changed, 19 insertions(+), 13 deletions(-)

diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index 6873c9b..b67b2ae 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -174,6 +174,7 @@ rpc_close_pipes(struct inode *inode)
   pipe->ops->release_pipe(inode);
  cancel_delayed_work_sync(&pipe->queue_timeout);
  rpc_inode_setowner(inode, NULL);
+ RPC_I(inode)->pipe = NULL;
  mutex_unlock(&inode->i_mutex);
 }

@@ -203,12 +204,13 @@ rpc_destroy_inode(struct inode *inode)
 static int
 rpc_pipe_open(struct inode *inode, struct file *filp)
 {
- struct rpc_pipe *pipe = RPC_I(inode)->pipe;
+ struct rpc_pipe *pipe;
  int first_open;
  int res = -ENXIO;

  mutex_lock(&inode->i_mutex);

```
- if (pipe->dentry == NULL)
+ pipe = RPC_I(inode)->pipe;
+ if (pipe == NULL)
    goto out;
  first_open = pipe->nreaders == 0 && pipe->nwriters == 0;
  if (first_open && pipe->ops->open_pipe) {
@@ -229,12 +231,13 @@ out:
 static int
 rpc_pipe_release(struct inode *inode, struct file *filp)
 {
- struct rpc_pipe *pipe = RPC_I(inode)->pipe;
+ struct rpc_pipe *pipe;
  struct rpc_pipe_msg *msg;
  int last_close;

  mutex_lock(&inode->i_mutex);
- if (pipe->dentry == NULL)
+ pipe = RPC_I(inode)->pipe;
+ if (pipe == NULL)
    goto out;
  msg = filp->private_data;
  if (msg != NULL) {
@@ -270,12 +273,13 @@ 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_pipe *pipe = RPC_I(inode)->pipe;
+ struct rpc_pipe *pipe;
  struct rpc_pipe_msg *msg;
  int res = 0;

  mutex_lock(&inode->i_mutex);
- if (pipe->dentry == NULL) {
+ pipe = RPC_I(inode)->pipe;
+ if (pipe == NULL) {
    res = -EPIPE;
    goto out_unlock;
  }
@@ -313,13 +317,12 @@ 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_pipe *pipe = RPC_I(inode)->pipe;
  int res;

  mutex_lock(&inode->i_mutex);
  res = -EPIPE;
- if (pipe->dentry != NULL)
```

```
-  res = pipe->ops->downcall(filp, buf, len);
+ if (RPC_I(inode)->pipe != NULL)
+  res = RPC_I(inode)->pipe->ops->downcall(filp, buf, len);
  mutex_unlock(&inode->i_mutex);
  return res;
 }
@@ -344,16 +347,18 @@ 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_pipe *pipe = RPC_I(inode)->pipe;
+ struct rpc_pipe *pipe;
  int len;

  switch (cmd) {
  case FIONREAD:
-  spin_lock(&pipe->lock);
-  if (pipe->dentry == NULL) {
-   spin_unlock(&pipe->lock);
+  mutex_lock(&inode->i_mutex);
+  pipe = RPC_I(inode)->pipe;
+  if (pipe == NULL) {
+   mutex_unlock(&inode->i_mutex);
    return -EPIPE;
   }
+  spin_lock(&pipe->lock);
   len = pipe->pipelen;
   if (filp->private_data) {
    struct rpc_pipe_msg *msg;
@@ -361,6 +366,7 @@ rpc_pipe_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
    len += msg->len - msg->copied;
   }
   spin_unlock(&pipe->lock);
+  mutex_unlock(&inode->i_mutex);
   return put_user(len, (int __user *)arg);
  default:
   return -EINVAL;
```

---

## Subject: [PATCH v2 4/4] SUNRPC: move waitq from RPC pipe to RPC inode
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 15:51:20 GMT
View Forum Message <> Reply to Message

Currently, wait queue, used for polling of RPC pipe changes from user-space,
is a part of RPC pipe. But the pipe data itself can be released on NFS umount
prior to dentry-inode pair, connected to it (is case of this pair is open by
some process).
This is not a problem for almost all pipe users, because all PipeFS file

operations checks pipe reference prior to using it.
Except evenfd. This thing registers itself with "poll" file operation and thus has a reference to pipe wait queue. This leads to oopses on destroying eventfd after NFS umount (like rpc_idmapd do) since not pipe data left to the point already.
The solution is to wait queue from pipe data to internal RPC inode data. This looks more logical, because this wiat queue used only for user-space processes, which already holds inode reference.

Note: upcalls have to get pipe->dentry prior to dereferecing wait queue to make sure, that mount point won't disappear from underneath us.

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

```
---
 include/linux/sunrpc/rpc_pipe_fs.h |    2 +-
 net/sunrpc/rpc_pipe.c              |   39 ++++++++++++++++++++++++------------
 2 files changed, 27 insertions(+), 14 deletions(-)

diff --git a/include/linux/sunrpc/rpc_pipe_fs.h b/include/linux/sunrpc/rpc_pipe_fs.h
index 426ce6e..a7b422b 100644
--- a/include/linux/sunrpc/rpc_pipe_fs.h
+++ b/include/linux/sunrpc/rpc_pipe_fs.h
@@ -28,7 +28,6 @@ struct rpc_pipe {
  int pipelen;
  int nreaders;
  int nwriters;
- wait_queue_head_t waitq;
 #define RPC_PIPE_WAIT_FOR_OPEN 1
  int flags;
  struct delayed_work queue_timeout;
@@ -41,6 +40,7 @@ struct rpc_inode {
  struct inode vfs_inode;
  void *private;
  struct rpc_pipe *pipe;
+ wait_queue_head_t waitq;
 };

 static inline struct rpc_inode *
diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index b67b2ae..ac9ee15 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -57,7 +57,7 @@ void rpc_pipefs_notifier_unregister(struct notifier_block *nb)
 }
 EXPORT_SYMBOL_GPL(rpc_pipefs_notifier_unregister);

-static void rpc_purge_list(struct rpc_pipe *pipe, struct list_head *head,
```

```
+static void rpc_purge_list(wait_queue_head_t *waitq, struct list_head *head,
   void (*destroy_msg)(struct rpc_pipe_msg *), int err)
 {
  struct rpc_pipe_msg *msg;
@@ -70,7 +70,7 @@ static void rpc_purge_list(struct rpc_pipe *pipe, struct list_head *head,
   msg->errno = err;
   destroy_msg(msg);
  } while (!list_empty(head));
- wake_up(&pipe->waitq);
+ wake_up(waitq);
 }

 static void
@@ -80,6 +80,7 @@ rpc_timeout_upcall_queue(struct work_struct *work)
  struct rpc_pipe *pipe =
   container_of(work, struct rpc_pipe, queue_timeout.work);
  void (*destroy_msg)(struct rpc_pipe_msg *);
+ struct dentry *dentry;

  spin_lock(&pipe->lock);
  destroy_msg = pipe->ops->destroy_msg;
@@ -87,8 +88,13 @@ rpc_timeout_upcall_queue(struct work_struct *work)
   list_splice_init(&pipe->pipe, &free_list);
   pipe->pipelen = 0;
  }
+ dentry = dget(pipe->dentry);
  spin_unlock(&pipe->lock);
- rpc_purge_list(pipe, &free_list, destroy_msg, -ETIMEDOUT);
+ if (dentry) {
+  rpc_purge_list(&RPC_I(dentry->d_inode)->waitq,
+         &free_list, destroy_msg, -ETIMEDOUT);
+  dput(dentry);
+ }
 }

 ssize_t rpc_pipe_generic_upcall(struct file *filp, struct rpc_pipe_msg *msg,
@@ -125,6 +131,7 @@ int
 rpc_queue_upcall(struct rpc_pipe *pipe, struct rpc_pipe_msg *msg)
 {
  int res = -EPIPE;
+ struct dentry *dentry;

  spin_lock(&pipe->lock);
  if (pipe->nreaders) {
@@ -140,8 +147,12 @@ rpc_queue_upcall(struct rpc_pipe *pipe, struct rpc_pipe_msg *msg)
   pipe->pipelen += msg->len;
   res = 0;
  }
```

```
+ dentry = dget(pipe->dentry);
  spin_unlock(&pipe->lock);
- wake_up(&pipe->waitq);
+ if (dentry) {
+  wake_up(&RPC_I(dentry->d_inode)->waitq);
+  dput(dentry);
+ }
  return res;
 }
 EXPORT_SYMBOL_GPL(rpc_queue_upcall);
@@ -168,7 +179,7 @@ rpc_close_pipes(struct inode *inode)
  pipe->pipelen = 0;
  pipe->dentry = NULL;
  spin_unlock(&pipe->lock);
- rpc_purge_list(pipe, &free_list, pipe->ops->destroy_msg, -EPIPE);
+ rpc_purge_list(&RPC_I(inode)->waitq, &free_list, pipe->ops->destroy_msg, -EPIPE);
  pipe->nwriters = 0;
  if (need_release && pipe->ops->release_pipe)
   pipe->ops->release_pipe(inode);
@@ -257,7 +268,7 @@ rpc_pipe_release(struct inode *inode, struct file *filp)
    list_splice_init(&pipe->pipe, &free_list);
    pipe->pipelen = 0;
    spin_unlock(&pipe->lock);
-   rpc_purge_list(pipe, &free_list,
+   rpc_purge_list(&RPC_I(inode)->waitq, &free_list,
      pipe->ops->destroy_msg, -EAGAIN);
   }
  }
@@ -330,16 +341,18 @@ rpc_pipe_write(struct file *filp, const char __user *buf, size_t len, loff_t
*of
 static unsigned int
 rpc_pipe_poll(struct file *filp, struct poll_table_struct *wait)
 {
- struct rpc_pipe *pipe = RPC_I(filp->f_path.dentry->d_inode)->pipe;
- unsigned int mask = 0;
+ struct inode *inode = filp->f_path.dentry->d_inode;
+ struct rpc_inode *rpci = RPC_I(inode);
+ unsigned int mask = POLLOUT | POLLWRNORM;

- poll_wait(filp, &pipe->waitq, wait);
+ poll_wait(filp, &rpci->waitq, wait);

- mask = POLLOUT | POLLWRNORM;
- if (pipe->dentry == NULL)
+ mutex_lock(&inode->i_mutex);
+ if (rpci->pipe == NULL)
   mask |= POLLERR | POLLHUP;
- if (filp->private_data || !list_empty(&pipe->pipe))
```

```
+ else if (filp->private_data || !list_empty(&rpci->pipe->pipe))
   mask |= POLLIN | POLLRDNORM;
+ mutex_unlock(&inode->i_mutex);
  return mask;
 }

@@ -543,7 +556,6 @@ init_pipe(struct rpc_pipe *pipe)
  INIT_LIST_HEAD(&pipe->in_downcall);
  INIT_LIST_HEAD(&pipe->pipe);
  pipe->pipelen = 0;
- init_waitqueue_head(&pipe->waitq);
  INIT_DELAYED_WORK(&pipe->queue_timeout,
        rpc_timeout_upcall_queue);
  pipe->ops = NULL;
@@ -1165,6 +1177,7 @@ init_once(void *foo)
  inode_init_once(&rpci->vfs_inode);
  rpci->private = NULL;
  rpci->pipe = NULL;
+ init_waitqueue_head(&rpci->waitq);
 }

 int register_rpc_pipefs(void)
```

---

Subject: RE: [PATCH v2 2/4] NFS: release per-net clients lock before calling PipeFS dentries creation
Posted by David Laight on Mon, 27 Feb 2012 15:59:23 GMT
View Forum Message <> Reply to Message

```
>   spin_lock(&nn->nfs_client_lock);
> - list_for_each_entry(clp, &nn->nfs_client_list, cl_share_link) {
> + list_for_each_entry_safe(clp, tmp, &nn->nfs_client_list,
cl_share_link) {
>    if (clp->rpc_ops != &nfs_v4_clientops)
>     continue;
> +  atomic_inc(&clp->cl_count);
> +  spin_unlock(&nn->nfs_client_lock);
>    error = __rpc_pipefs_event(clp, event, sb);
> +  nfs_put_client(clp);
>    if (error)
>     break;
> +  spin_lock(&nn->nfs_client_lock);
>   }
>   spin_unlock(&nn->nfs_client_lock);
>   return error;
```

The locking doesn't look right if the loop breaks on error.
(Same applied to patch v2 1/4)

Although list_fo_each_entry_safe() allows the current entry
to be freed, I don't believe it allows the 'next' to be freed.
I doubt there is protection against that happening.

Do you need to use an atomic_inc() for cl_count.
I'd guess the nfs_client_lock is usually held?

David

---

Subject: Re: [PATCH v2 2/4] NFS: release per-net clients lock before calling PipeFS dentries creation
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 16:20:46 GMT
View Forum Message <> Reply to Message

>
>> spin_lock(&nn->nfs_client_lock);
>> - list_for_each_entry(clp,&nn->nfs_client_list, cl_share_link) {
>> + list_for_each_entry_safe(clp, tmp,&nn->nfs_client_list,
> cl_share_link) {
>> if (clp->rpc_ops !=&nfs_v4_clientops)
>> continue;
>> + atomic_inc(&clp->cl_count);
>> + spin_unlock(&nn->nfs_client_lock);
>> error = __rpc_pipefs_event(clp, event, sb);
>> + nfs_put_client(clp);
>> if (error)
>> break;
>> + spin_lock(&nn->nfs_client_lock);
>> }
>> spin_unlock(&nn->nfs_client_lock);
>> return error;
>
> The locking doesn't look right if the loop breaks on error.
> (Same applied to patch v2 1/4)
>

Thanks for the catch. I'll fix this.

> Although list_fo_each_entry_safe() allows the current entry
> to be freed, I don't believe it allows the 'next' to be freed.
> I doubt there is protection against that happening.
>

We need to use safe macro, because client can be destroyed on nfs_put_client() call.
About "protection against ... the 'next' to be freed" - I dont' think, that we

need any protection against it. This will be done under nfs_client_lock, and
current entry list pointers will be updated properly.

> Do you need to use an atomic_inc() for cl_count.
> I'd guess the nfs_client_lock is usually held?
>

Sorry, I don't understand this question.

--
Best regards,
Stanislav Kinsbursky

---

Subject: Re: [PATCH v2 1/4] SUNRPC: release per-net clients lock before calling PipeFS dentries creation
Posted by Myklebust, Trond on Mon, 27 Feb 2012 16:21:32 GMT
View Forum Message <> Reply to Message

On Mon, 2012-02-27 at 19:50 +0400, Stanislav Kinsbursky wrote:
> Lockdep is sad otherwise, because inode mutex is taken on PipeFS dentry
> creation, which can be called on mount notification, where this per-net client
> lock is taken on clients list walk.
>
> Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>
>
> ---
>  net/sunrpc/clnt.c |   10 +++++++---
>  1 files changed, 7 insertions(+), 3 deletions(-)
>
> diff --git a/net/sunrpc/clnt.c b/net/sunrpc/clnt.c
> index bb7ed2f3..ddb5741 100644
> --- a/net/sunrpc/clnt.c
> +++ b/net/sunrpc/clnt.c
> @@ -84,7 +84,7 @@ static void rpc_register_client(struct rpc_clnt *clnt)
>   struct sunrpc_net *sn = net_generic(clnt->cl_xprt->xprt_net, sunrpc_net_id);
>
>   spin_lock(&sn->rpc_client_lock);
> - list_add(&clnt->cl_clients, &sn->all_clients);
> + list_add_tail(&clnt->cl_clients, &sn->all_clients);
>   spin_unlock(&sn->rpc_client_lock);
>  }
>
> @@ -208,15 +208,19 @@ static int rpc_pipefs_event(struct notifier_block *nb, unsigned long
event,
>         void *ptr)
>  {
>   struct super_block *sb = ptr;

> - struct rpc_clnt *clnt;
> + struct rpc_clnt *clnt, *tmp;
>   int error = 0;
>   struct sunrpc_net *sn = net_generic(sb->s_fs_info, sunrpc_net_id);
>
>   spin_lock(&sn->rpc_client_lock);
> - list_for_each_entry(clnt, &sn->all_clients, cl_clients) {
> + list_for_each_entry_safe(clnt, tmp, &sn->all_clients, cl_clients) {
> +  atomic_inc(&clnt->cl_count);
> +  spin_unlock(&sn->rpc_client_lock);
>    error = __rpc_pipefs_event(clnt, event, sb);
> +  rpc_release_client(clnt);
>    if (error)
>     break;
> +  spin_lock(&sn->rpc_client_lock);
>   }
>   spin_unlock(&sn->rpc_client_lock);
>   return error;
>

This won't be safe. Nothing guarantees that 'tmp' remains valid after
you drop the spin_lock.

I think you rather need to add a check for whether clnt->cl_dentry is in
the right state (NULL if RPC_PIPEFS_UMOUNT or non-NULL if
RPC_PIPEFS_MOUNT) before deciding whether or not to atomic_inc() and
drop the lock, so that you can restart the loop after calling
__rpc_pipefs_event().

--
Trond Myklebust
Linux NFS client maintainer

NetApp
Trond.Myklebust@netapp.com
www.netapp.com

---

Subject: Re: [PATCH v2 1/4] SUNRPC: release per-net clients lock before calling PipeFS dentries creation
Posted by Stanislav Kinsbursky on Mon, 27 Feb 2012 16:55:12 GMT
View Forum Message <> Reply to Message

> On Mon, 2012-02-27 at 19:50 +0400, Stanislav Kinsbursky wrote:
>> Lockdep is sad otherwise, because inode mutex is taken on PipeFS dentry
>> creation, which can be called on mount notification, where this per-net client

>> lock is taken on clients list walk.
>>
>> Signed-off-by: Stanislav Kinsbursky<skinsbursky@parallels.com>
>>
>> ---
>>  net/sunrpc/clnt.c |  10 +++++++---
>>  1 files changed, 7 insertions(+), 3 deletions(-)
>>
>> diff --git a/net/sunrpc/clnt.c b/net/sunrpc/clnt.c
>> index bb7ed2f3..ddb5741 100644
>> --- a/net/sunrpc/clnt.c
>> +++ b/net/sunrpc/clnt.c
>> @@ -84,7 +84,7 @@ static void rpc_register_client(struct rpc_clnt *clnt)
>>   struct sunrpc_net *sn = net_generic(clnt->cl_xprt->xprt_net, sunrpc_net_id);
>>
>>   spin_lock(&sn->rpc_client_lock);
>> - list_add(&clnt->cl_clients,&sn->all_clients);
>> + list_add_tail(&clnt->cl_clients,&sn->all_clients);
>>   spin_unlock(&sn->rpc_client_lock);
>>  }
>>
>> @@ -208,15 +208,19 @@ static int rpc_pipefs_event(struct notifier_block *nb, unsigned long
event,
>>         void *ptr)
>>  {
>>   struct super_block *sb = ptr;
>> - struct rpc_clnt *clnt;
>> + struct rpc_clnt *clnt, *tmp;
>>   int error = 0;
>>   struct sunrpc_net *sn = net_generic(sb->s_fs_info, sunrpc_net_id);
>>
>>   spin_lock(&sn->rpc_client_lock);
>> - list_for_each_entry(clnt,&sn->all_clients, cl_clients) {
>> + list_for_each_entry_safe(clnt, tmp,&sn->all_clients, cl_clients) {
>> +  atomic_inc(&clnt->cl_count);
>> +  spin_unlock(&sn->rpc_client_lock);
>>    error = __rpc_pipefs_event(clnt, event, sb);
>> +  rpc_release_client(clnt);
>>    if (error)
>>     break;
>> +  spin_lock(&sn->rpc_client_lock);
>>   }
>>   spin_unlock(&sn->rpc_client_lock);
>>   return error;
>>
>
> This won't be safe. Nothing guarantees that 'tmp' remains valid after
> you drop the spin_lock.

>
> I think you rather need to add a check for whether clnt->cl_dentry is in
> the right state (NULL if RPC_PIPEFS_UMOUNT or non-NULL if
> RPC_PIPEFS_MOUNT) before deciding whether or not to atomic_inc() and
> drop the lock, so that you can restart the loop after calling
> __rpc_pipefs_event().
>

Gmmm.
Please, correct me, if I'm wrong, that you are proposing something like this:

```
    spin_lock(&sn->rpc_client_lock);
again:
 list_for_each_entry(clnt,&sn->all_clients, cl_clients) {
  if ((event == RPC_PIPEFS_MOUNT) && clnt->cl_dentry) ||
      (event == RPC_PIPEFS_UMOUNT) && !clnt->cl_dentry))
   continue;
  atomic_inc(&clnt->cl_count);
  spin_unlock(&sn->rpc_client_lock);
      error = __rpc_pipefs_event(clnt, event, sb);
  rpc_release_client(clnt);
  spin_lock(&sn->rpc_client_lock);
      if (error)
       break;
  goto again;
     }
     spin_unlock(&sn->rpc_client_lock);
```

--
Best regards,
Stanislav Kinsbursky

---

Subject: RE: [PATCH v2 1/4] SUNRPC: release per-net clients lock before calling PipeFS dentries creation
Posted by David Laight on Mon, 27 Feb 2012 17:11:37 GMT
View Forum Message <> Reply to Message

> Gmmm.
> Please, correct me, if I'm wrong, that you are proposing
> something like this:
>
>     spin_lock(&sn->rpc_client_lock);
> again:
>  list_for_each_entry(clnt,&sn->all_clients, cl_clients) {
>   if ((event == RPC_PIPEFS_MOUNT) && clnt->cl_dentry) ||
>       (event == RPC_PIPEFS_UMOUNT) && !clnt->cl_dentry))

```
>    continue;
>   atomic_inc(&clnt->cl_count);
>   spin_unlock(&sn->rpc_client_lock);
>       error = __rpc_pipefs_event(clnt, event, sb);
>   rpc_release_client(clnt);
>   spin_lock(&sn->rpc_client_lock);
>       if (error)
>        break;
>   goto again;
>      }
>      spin_unlock(&sn->rpc_client_lock);
```

You might manage to request reference to the 'next'
item while holding the lock. So maybe a loop that has:

```
 error = ...
 if (error) {
  rpc_release_client(clnt);
  break;
 }
 spin_lock(&sn->rpc_client_lock);
 next_clnt = list_next(clnt);
 if (next_clnt)
  atomic_inc(&next_clnt->cl_count);
 spin_unlock(&sn->rpc_client_lock);
 rpc_release_client(clnt);
 clnt = next_clnt;
   } while (clnt != NULL);
```

David

---

Subject: Re: [PATCH v2 1/4] SUNRPC: release per-net clients lock before calling PipeFS dentries creation
Posted by Myklebust, Trond on Mon, 27 Feb 2012 17:37:13 GMT
View Forum Message <> Reply to Message

On Mon, 2012-02-27 at 20:55 +0400, Stanislav Kinsbursky wrote:

> > On Mon, 2012-02-27 at 19:50 +0400, Stanislav Kinsbursky wrote:
> >> Lockdep is sad otherwise, because inode mutex is taken on PipeFS dentry
> >> creation, which can be called on mount notification, where this per-net client
> >> lock is taken on clients list walk.
> >>
> >> Signed-off-by: Stanislav Kinsbursky<skinsbursky@parallels.com>
> >>
> >> ---
> >>  net/sunrpc/clnt.c |  10 +++++++---

> >>   1 files changed, 7 insertions(+), 3 deletions(-)
> >>
> >> diff --git a/net/sunrpc/clnt.c b/net/sunrpc/clnt.c
> >> index bb7ed2f3..ddb5741 100644
> >> --- a/net/sunrpc/clnt.c
> >> +++ b/net/sunrpc/clnt.c
> >> @@ -84,7 +84,7 @@ static void rpc_register_client(struct rpc_clnt *clnt)
> >>    struct sunrpc_net *sn = net_generic(clnt->cl_xprt->xprt_net, sunrpc_net_id);
> >>
> >>    spin_lock(&sn->rpc_client_lock);
> >> - list_add(&clnt->cl_clients,&sn->all_clients);
> >> + list_add_tail(&clnt->cl_clients,&sn->all_clients);
> >>    spin_unlock(&sn->rpc_client_lock);
> >>   }
> >>
> >> @@ -208,15 +208,19 @@ static int rpc_pipefs_event(struct notifier_block *nb, unsigned
long event,
> >>          void *ptr)
> >>   {
> >>    struct super_block *sb = ptr;
> >> - struct rpc_clnt *clnt;
> >> + struct rpc_clnt *clnt, *tmp;
> >>    int error = 0;
> >>    struct sunrpc_net *sn = net_generic(sb->s_fs_info, sunrpc_net_id);
> >>
> >>    spin_lock(&sn->rpc_client_lock);
> >> - list_for_each_entry(clnt,&sn->all_clients, cl_clients) {
> >> + list_for_each_entry_safe(clnt, tmp,&sn->all_clients, cl_clients) {
> >> +  atomic_inc(&clnt->cl_count);
> >> +  spin_unlock(&sn->rpc_client_lock);
> >>     error = __rpc_pipefs_event(clnt, event, sb);
> >> +  rpc_release_client(clnt);
> >>     if (error)
> >>      break;
> >> +  spin_lock(&sn->rpc_client_lock);
> >>    }
> >>    spin_unlock(&sn->rpc_client_lock);
> >>    return error;
> >>
> >
> > This won't be safe. Nothing guarantees that 'tmp' remains valid after
> > you drop the spin_lock.
> >
> > I think you rather need to add a check for whether clnt->cl_dentry is in
> > the right state (NULL if RPC_PIPEFS_UMOUNT or non-NULL if
> > RPC_PIPEFS_MOUNT) before deciding whether or not to atomic_inc() and
> > drop the lock, so that you can restart the loop after calling
> > __rpc_pipefs_event().

> >
>
> Gmmm.
> Please, correct me, if I'm wrong, that you are proposing something like this:
>
>     spin_lock(&sn->rpc_client_lock);
> again:
>  list_for_each_entry(clnt,&sn->all_clients, cl_clients) {
>   if ((event == RPC_PIPEFS_MOUNT) && clnt->cl_dentry) ||
>       (event == RPC_PIPEFS_UMOUNT) && !clnt->cl_dentry))
>    continue;
>   atomic_inc(&clnt->cl_count);
>   spin_unlock(&sn->rpc_client_lock);
>       error = __rpc_pipefs_event(clnt, event, sb);
>   rpc_release_client(clnt);
>   spin_lock(&sn->rpc_client_lock);
>       if (error)
>        break;
>   goto again;
>      }
>      spin_unlock(&sn->rpc_client_lock);

Something along those lines, yes... Alternatively, you could do as David
proposes, and grab a reference to the next rpc_client before calling
rpc_release_client. Either works for me...

--
Trond Myklebust
Linux NFS client maintainer

NetApp
Trond.Myklebust@netapp.com
www.netapp.com