Subject: [PATCH 0/6] NFS: create clients and IDMAP pipes per network namespace
Posted by Stanislav Kinsbursky on Mon, 28 Nov 2011 13:36:11 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: v3.1

This patch set depends on previous patch sets titled:
1) "SUNRPC: initial part of making pipefs work in net ns"
2) "SUNPRC: cleanup PipeFS for network-namespace-aware users"
3) "SUNRPC: make RPC clients use network-namespace-aware PipeFS routines"

Actually, this patch set consists of two tightly connected parts:
1) NFS clients creation per network namespace
2) IDMAP pipe dentries creation in owner client network namesapce context

This patch set is the third part of making "PipeFS per network namespace".

The following series consists of:

Stanislav Kinsbursky (6):
    SUNRPC: fix pipe->ops cleanup on pipe dentry unlink
    NFS: make NFS client allocated per network namespace context
    NFS: pass NFS client owner network namespace to RPC client creation routine
    NFS: create callback transports in parent transport network namespace
    NFS: handle NFS idmap pipe PipeFS dentries by network namespace aware routines
    NFS: idmap PipeFS notifier introduced

```
 fs/nfs/callback.c                |   10 +--
 fs/nfs/client.c                  |   22 ++++--
 fs/nfs/idmap.c                   |  136 ++++++++++++++++++++++++++++++++++++--
 fs/nfs/internal.h                |    6 ++
 fs/nfs/mount_clnt.c              |    4 +
 fs/nfs/super.c                   |    4 +
 include/linux/nfs_fs_sb.h        |    1
 include/linux/nfs_idmap.h        |   13 +--
 include/linux/sunrpc/rpc_pipe_fs.h |    7 ++
 net/sunrpc/clnt.c                |    1
 net/sunrpc/rpc_pipe.c            |   67 ++++++++++++--------
 11 files changed, 208 insertions(+), 63 deletions(-)
```

--
Signature

Subject: [PATCH 1/6] SUNRPC: fix pipe->ops cleanup on pipe dentry unlink
Posted by Stanislav Kinsbursky on Mon, 28 Nov 2011 13:38:37 GMT
View Forum Message <> Reply to Message

This patch looks late due to GSS AUTH patches sent already. But it fixes a flaw
in RPC PipeFS pipes handling.
I've added this patch in the series, because this series related to pipes. But
it should be a part of previous series named "SUNPRC: cleanup PipeFS for
network-namespace-aware users".

Pipe dentry can be created and destroyed many times during pipe life cycle.
This actually means, that we can't set pipe->ops to NULL in rpc_close_pipes()
and use this variable as a flag, indicating, that pipe's dentry is unlinking.
To follow this restriction, this patch replaces "pipe->ops = NULL" assignment
and checks for NULL with "pipe->dentry = NULL" assignment and checks for
NULL respectively.
This patch also removes check for non-NULL pipe->ops (or pipe->dentry) in
rpc_close_pipes() because it always non-NULL now.

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

---
 net/sunrpc/rpc_pipe.c | 51 ++++++++++++++++++++----------------------
 1 files changed, 20 insertions(+), 31 deletions(-)

diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index 1ea0dcf..b80d7bd 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -86,10 +86,6 @@ rpc_timeout_upcall_queue(struct work_struct *work)
  void (*destroy_msg)(struct rpc_pipe_msg *);

  spin_lock(&pipe->lock);
- if (pipe->ops == NULL) {
- spin_unlock(&pipe->lock);
- return;
- }
  destroy_msg = pipe->ops->destroy_msg;
  if (pipe->nreaders == 0) {
   list_splice_init(&pipe->pipe, &free_list);
@@ -115,8 +111,6 @@ rpc_queue_upcall(struct rpc_pipe *pipe, struct rpc_pipe_msg *msg)
  int res = -EPIPE;

  spin_lock(&pipe->lock);
- if (pipe->ops == NULL)
- goto out;
  if (pipe->nreaders) {
   list_add_tail(&msg->list, &pipe->pipe);
   pipe->pipelen += msg->len;

```
@@ -130,7 +124,6 @@ rpc_queue_upcall(struct rpc_pipe *pipe, struct rpc_pipe_msg *msg)
   pipe->pipelen += msg->len;
   res = 0;
  }
-out:
  spin_unlock(&pipe->lock);
  wake_up(&pipe->waitq);
  return res;
@@ -147,27 +140,23 @@ static void
 rpc_close_pipes(struct inode *inode)
 {
  struct rpc_pipe *pipe = RPC_I(inode)->pipe;
- const struct rpc_pipe_ops *ops;
  int need_release;
+ LIST_HEAD(free_list);

  mutex_lock(&inode->i_mutex);
- ops = pipe->ops;
- if (ops != NULL) {
-  LIST_HEAD(free_list);
-  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(&pipe->queue_timeout);
- }
+ 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->dentry = NULL;
+ spin_unlock(&pipe->lock);
+ rpc_purge_list(pipe, &free_list, pipe->ops->destroy_msg, -EPIPE);
+ pipe->nwriters = 0;
+ if (need_release && pipe->ops->release_pipe)
+  pipe->ops->release_pipe(inode);
+ cancel_delayed_work_sync(&pipe->queue_timeout);
  rpc_inode_setowner(inode, NULL);
```

```
  mutex_unlock(&inode->i_mutex);
 }
@@ -204,7 +193,7 @@ rpc_pipe_open(struct inode *inode, struct file *filp)
  int res = -ENXIO;

  mutex_lock(&inode->i_mutex);
- if (pipe->ops == NULL)
+ if (pipe->dentry == NULL)
   goto out;
  first_open = pipe->nreaders == 0 && pipe->nwriters == 0;
  if (first_open && pipe->ops->open_pipe) {
@@ -230,7 +219,7 @@ rpc_pipe_release(struct inode *inode, struct file *filp)
  int last_close;

  mutex_lock(&inode->i_mutex);
- if (pipe->ops == NULL)
+ if (pipe->dentry == NULL)
   goto out;
  msg = filp->private_data;
  if (msg != NULL) {
@@ -271,7 +260,7 @@ rpc_pipe_read(struct file *filp, char __user *buf, size_t len, loff_t *offset)
  int res = 0;

  mutex_lock(&inode->i_mutex);
- if (pipe->ops == NULL) {
+ if (pipe->dentry == NULL) {
   res = -EPIPE;
   goto out_unlock;
  }
@@ -314,7 +303,7 @@ rpc_pipe_write(struct file *filp, const char __user *buf, size_t len, loff_t *of

  mutex_lock(&inode->i_mutex);
  res = -EPIPE;
- if (pipe->ops != NULL)
+ if (pipe->dentry != NULL)
   res = pipe->ops->downcall(filp, buf, len);
  mutex_unlock(&inode->i_mutex);
  return res;
@@ -329,7 +318,7 @@ rpc_pipe_poll(struct file *filp, struct poll_table_struct *wait)
  poll_wait(filp, &pipe->waitq, wait);

  mask = POLLOUT | POLLWRNORM;
- if (pipe->ops == NULL)
+ if (pipe->dentry == NULL)
   mask |= POLLERR | POLLHUP;
  if (filp->private_data || !list_empty(&pipe->pipe))
   mask |= POLLIN | POLLRDNORM;
@@ -346,7 +335,7 @@ rpc_pipe_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
```

```
  switch (cmd) {
  case FIONREAD:
   spin_lock(&pipe->lock);
-  if (pipe->ops == NULL) {
+  if (pipe->dentry == NULL) {
    spin_unlock(&pipe->lock);
    return -EPIPE;
   }
```

Subject: [PATCH 2/6] NFS: make NFS client allocated per network namespace context
Posted by Stanislav Kinsbursky on Mon, 28 Nov 2011 13:38:38 GMT
View Forum Message <> Reply to Message

This patch adds new net variable to nfs_client structure. This variable is set
on NFS client creation and cheched during matching NFS client search.
Initially current->nsproxy->net_ns is used as network namespace owner for new
NFS client to create. This network namespace pointer is set during mount
options parsing and thus can be passed from user-spave utils in future if will
be necessary.

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

```
---
 fs/nfs/client.c          |   16 +++++++++++++---
 fs/nfs/internal.h        |    1 +
 fs/nfs/super.c           |    3 +++
 include/linux/nfs_fs_sb.h |    1 +
 4 files changed, 18 insertions(+), 3 deletions(-)

diff --git a/fs/nfs/client.c b/fs/nfs/client.c
index 5833fbb..47a8cd6 100644
--- a/fs/nfs/client.c
+++ b/fs/nfs/client.c
@@ -135,6 +135,7 @@ struct nfs_client_initdata {
  const struct nfs_rpc_ops *rpc_ops;
  int proto;
  u32 minorversion;
+ struct net *net;
 };

 /*
@@ -189,6 +190,7 @@ static struct nfs_client *nfs_alloc_client(const struct nfs_client_initdata
*cl_
  if (!IS_ERR(cred))
   clp->cl_machine_cred = cred;
  nfs_fscache_get_client_cookie(clp);
```

```
+ clp->net = cl_init->net;

  return clp;

@@ -475,6 +477,9 @@ static struct nfs_client *nfs_match_client(const struct nfs_client_initdata
*dat
    /* Match the full socket address */
    if (!nfs_sockaddr_cmp(sap, clap))
     continue;
+   /* Match network namespace */
+   if (clp->net != data->net)
+    continue;

    atomic_inc(&clp->cl_count);
    return clp;
@@ -825,6 +830,7 @@ static int nfs_init_server(struct nfs_server *server,
    .addrlen = data->nfs_server.addrlen,
    .rpc_ops = &nfs_v2_clientops,
    .proto = data->nfs_server.protocol,
+   .net = data->net,
   };
   struct rpc_timeout timeparms;
   struct nfs_client *clp;
@@ -1386,7 +1392,7 @@ static int nfs4_set_client(struct nfs_server *server,
    const char *ip_addr,
    rpc_authflavor_t authflavour,
    int proto, const struct rpc_timeout *timeparms,
-   u32 minorversion)
+   u32 minorversion, struct net *net)
  {
   struct nfs_client_initdata cl_init = {
    .hostname = hostname,
@@ -1395,6 +1401,7 @@ static int nfs4_set_client(struct nfs_server *server,
    .rpc_ops = &nfs_v4_clientops,
    .proto = proto,
    .minorversion = minorversion,
+   .net = net,
   };
   struct nfs_client *clp;
   int error;
@@ -1446,6 +1453,7 @@ struct nfs_client *nfs4_set_ds_client(struct nfs_client* mds_clp,
    .rpc_ops = &nfs_v4_clientops,
    .proto = ds_proto,
    .minorversion = mds_clp->cl_minorversion,
+   .net = mds_clp->net,
   };
   struct rpc_timeout ds_timeout = {
    .to_initval = 15 * HZ,
```

```
@@ -1572,7 +1580,8 @@ static int nfs4_init_server(struct nfs_server *server,
    data->auth_flavors[0],
    data->nfs_server.protocol,
    &timeparms,
-   data->minorversion);
+   data->minorversion,
+   data->net);
  if (error < 0)
   goto error;

@@ -1669,7 +1678,8 @@ struct nfs_server *nfs4_create_referral_server(struct nfs_clone_mount
*data,
     data->authflavor,
     parent_server->client->cl_xprt->prot,
     parent_server->client->cl_timeout,
-    parent_client->cl_mvops->minor_version);
+    parent_client->cl_mvops->minor_version,
+    parent_client->net);
  if (error < 0)
   goto error;

diff --git a/fs/nfs/internal.h b/fs/nfs/internal.h
index ab12913..4565e76 100644
--- a/fs/nfs/internal.h
+++ b/fs/nfs/internal.h
@@ -123,6 +123,7 @@ struct nfs_parsed_mount_data {
  } nfs_server;

  struct security_mnt_opts lsm_opts;
+ struct net  *net;
 };

 /* mount_clnt.c */
diff --git a/fs/nfs/super.c b/fs/nfs/super.c
index 5b19b6a..782260d 100644
--- a/fs/nfs/super.c
+++ b/fs/nfs/super.c
@@ -53,6 +53,7 @@
 #include <linux/nfs_xdr.h>
 #include <linux/magic.h>
 #include <linux/parser.h>
+#include <linux/nsproxy.h>

 #include <asm/system.h>
 #include <asm/uaccess.h>
@@ -1089,6 +1090,8 @@ static int nfs_parse_mount_options(char *raw,

  free_secdata(secdata);
```

```
+ mnt->net = current->nsproxy->net_ns;
+
  while ((p = strsep(&raw, ",")) != NULL) {
   substring_t args[MAX_OPT_ARGS];
   unsigned long option;
diff --git a/include/linux/nfs_fs_sb.h b/include/linux/nfs_fs_sb.h
index b5479df..71fd605 100644
--- a/include/linux/nfs_fs_sb.h
+++ b/include/linux/nfs_fs_sb.h
@@ -85,6 +85,7 @@ struct nfs_client {
 #endif

  struct server_scope *server_scope; /* from exchange_id */
+ struct net  *net;
 };

 /*
```

Subject: [PATCH 5/6] NFS: handle NFS idmap pipe PipeFS dentries by network namespace aware routines

Posted by Stanislav Kinsbursky on Mon, 28 Nov 2011 13:38:41 GMT

View Forum Message <> Reply to Message

This patch makes NFS idmap pipes dentries allocated and destroyed in network
namespace context by PipeFS network namespace aware routines.
Network namespace context is obtained from nfs_client structure.

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

```
---
 fs/nfs/idmap.c |   61 ++++++++++++++++++++++++++++++++++++++++++++++++-------
 1 files changed, 53 insertions(+), 8 deletions(-)

diff --git a/fs/nfs/idmap.c b/fs/nfs/idmap.c
index 60698a1..0680f63 100644
--- a/fs/nfs/idmap.c
+++ b/fs/nfs/idmap.c
@@ -350,6 +350,56 @@ static const struct rpc_pipe_ops idmap_upcall_ops = {
  .destroy_msg = idmap_pipe_destroy_msg,
 };

+static void __nfs_idmap_unregister(struct rpc_pipe *pipe)
+{
+ if (pipe->dentry)
+  rpc_unlink(pipe->dentry);
+}
```

```
+
+static int __nfs_idmap_register(struct dentry *dir,
+         struct idmap *idmap,
+         struct rpc_pipe *pipe)
+{
+ struct dentry *dentry;
+
+ dentry = rpc_mkpipe_dentry(dir, "idmap", idmap, pipe);
+ if (IS_ERR(dentry))
+  return PTR_ERR(dentry);
+ pipe->dentry = dentry;
+ return 0;
+}
+
+static void nfs_idmap_unregister(struct nfs_client *clp,
+          struct rpc_pipe *pipe)
+{
+ struct net *net = clp->net;
+ struct super_block *pipefs_sb;
+
+ pipefs_sb = rpc_get_sb_net(net);
+ if (pipefs_sb) {
+  __nfs_idmap_unregister(pipe);
+  rpc_put_sb_net(net);
+ }
+}
+
+static int nfs_idmap_register(struct nfs_client *clp,
+       struct idmap *idmap,
+       struct rpc_pipe *pipe)
+{
+ struct net *net = clp->net;
+ struct super_block *pipefs_sb;
+ int err = 0;
+
+ pipefs_sb = rpc_get_sb_net(net);
+ if (pipefs_sb) {
+  if (clp->cl_rpcclient->cl_dentry)
+   err = __nfs_idmap_register(clp->cl_rpcclient->cl_dentry,
+         idmap, pipe);
+  rpc_put_sb_net(net);
+ }
+ return err;
+}
+
 int
 nfs_idmap_new(struct nfs_client *clp)
 {
```

```
@@ -369,12 +419,8 @@ nfs_idmap_new(struct nfs_client *clp)
   kfree(idmap);
   return error;
  }
-
- if (clp->cl_rpcclient->cl_dentry)
-  pipe->dentry = rpc_mkpipe_dentry(clp->cl_rpcclient->cl_dentry,
-    "idmap", idmap, pipe);
- if (IS_ERR(pipe->dentry)) {
-  error = PTR_ERR(pipe->dentry);
+ error = nfs_idmap_register(clp, idmap, pipe);
+ if (error) {
   rpc_destroy_pipe_data(pipe);
   kfree(idmap);
   return error;
@@ -397,8 +443,7 @@ nfs_idmap_delete(struct nfs_client *clp)

  if (!idmap)
   return;
- if (idmap->idmap_pipe->dentry)
-  rpc_unlink(idmap->idmap_pipe->dentry);
+ nfs_idmap_unregister(clp, idmap->idmap_pipe);
  rpc_destroy_pipe_data(idmap->idmap_pipe);
  clp->cl_idmap = NULL;
  kfree(idmap);
```

Subject: [PATCH 3/6] NFS: pass NFS client owner network namespace to RPC client creation routine
Posted by Stanislav Kinsbursky on Mon, 28 Nov 2011 13:38:43 GMT
View Forum Message <> Reply to Message

This patch replaces static "init_net" with nfs_client->net pointer in RPC
client creation calls.

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

```
---
 fs/nfs/client.c     |    2 +-
 fs/nfs/internal.h   |    1 +
 fs/nfs/mount_clnt.c |    4 ++--
 fs/nfs/super.c      |    1 +
 4 files changed, 5 insertions(+), 3 deletions(-)

diff --git a/fs/nfs/client.c b/fs/nfs/client.c
index 47a8cd6..58eebb5 100644
--- a/fs/nfs/client.c
+++ b/fs/nfs/client.c
```

```
@@ -641,7 +641,7 @@ static int nfs_create_rpc_client(struct nfs_client *clp,
 {
  struct rpc_clnt  *clnt = NULL;
  struct rpc_create_args args = {
-  .net  = &init_net,
+  .net  = clp->net,
    .protocol = clp->cl_proto,
    .address = (struct sockaddr *)&clp->cl_addr,
    .addrsize = clp->cl_addrlen,
diff --git a/fs/nfs/internal.h b/fs/nfs/internal.h
index 4565e76..451acb5 100644
--- a/fs/nfs/internal.h
+++ b/fs/nfs/internal.h
@@ -138,6 +138,7 @@ struct nfs_mount_request {
  int   noresvport;
  unsigned int  *auth_flav_len;
  rpc_authflavor_t *auth_flavs;
+ struct net  *net;
 };

 extern int nfs_mount(struct nfs_mount_request *info);
diff --git a/fs/nfs/mount_clnt.c b/fs/nfs/mount_clnt.c
index d4c2d6b..4fbe3a8 100644
--- a/fs/nfs/mount_clnt.c
+++ b/fs/nfs/mount_clnt.c
@@ -153,7 +153,7 @@ int nfs_mount(struct nfs_mount_request *info)
    .rpc_resp = &result,
  };
  struct rpc_create_args args = {
-  .net  = &init_net,
+  .net  = info->net,
    .protocol = info->protocol,
    .address = info->sap,
    .addrsize = info->salen,
@@ -225,7 +225,7 @@ void nfs_umount(const struct nfs_mount_request *info)
    .to_retries = 2,
  };
  struct rpc_create_args args = {
-  .net  = &init_net,
+  .net  = info->net,
    .protocol = IPPROTO_UDP,
    .address = info->sap,
    .addrsize = info->salen,
diff --git a/fs/nfs/super.c b/fs/nfs/super.c
index 782260d..b52819e 100644
--- a/fs/nfs/super.c
+++ b/fs/nfs/super.c
@@ -1607,6 +1607,7 @@ static int nfs_try_mount(struct nfs_parsed_mount_data *args,
```

```
   .noresvport = args->flags & NFS_MOUNT_NORESVPORT,
   .auth_flav_len = &server_authlist_len,
   .auth_flavs = server_authlist,
+  .net  = args->net,
  };
  int status;
```

Subject: [PATCH 4/6] NFS: create callback transports in parent transport network namespace
Posted by Stanislav Kinsbursky on Mon, 28 Nov 2011 13:38:45 GMT
View Forum Message <> Reply to Message

This patch replaces static "init_net" references with parent transport xprt_net
reference. Thus callback transports will be created in the same network
namespace as respective NFS mount point was created.

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

```
---
 fs/nfs/callback.c |   10 +++++-----
 1 files changed, 5 insertions(+), 5 deletions(-)

diff --git a/fs/nfs/callback.c b/fs/nfs/callback.c
index e3d2942..1d04f9d 100644
--- a/fs/nfs/callback.c
+++ b/fs/nfs/callback.c
@@ -102,11 +102,11 @@ nfs4_callback_svc(void *vrqstp)
  * Prepare to bring up the NFSv4 callback service
  */
 struct svc_rqst *
-nfs4_callback_up(struct svc_serv *serv)
+nfs4_callback_up(struct svc_serv *serv, struct rpc_xprt *xprt)
 {
  int ret;

- ret = svc_create_xprt(serv, "tcp", &init_net, PF_INET,
+ ret = svc_create_xprt(serv, "tcp", xprt->xprt_net, PF_INET,
      nfs_callback_set_tcpport, SVC_SOCK_ANONYMOUS);
  if (ret <= 0)
   goto out_err;
@@ -114,7 +114,7 @@ nfs4_callback_up(struct svc_serv *serv)
  dprintk("NFS: Callback listener port = %u (af %u)\n",
    nfs_callback_tcpport, PF_INET);

- ret = svc_create_xprt(serv, "tcp", &init_net, PF_INET6,
+ ret = svc_create_xprt(serv, "tcp", xprt->xprt_net, PF_INET6,
      nfs_callback_set_tcpport, SVC_SOCK_ANONYMOUS);
```

```
  if (ret > 0) {
   nfs_callback_tcpport6 = ret;
@@ -183,7 +183,7 @@ nfs41_callback_up(struct svc_serv *serv, struct rpc_xprt *xprt)
   * fore channel connection.
   * Returns the input port (0) and sets the svc_serv bc_xprt on success
   */
- ret = svc_create_xprt(serv, "tcp-bc", &init_net, PF_INET, 0,
+ ret = svc_create_xprt(serv, "tcp-bc", xprt->xprt_net, PF_INET, 0,
         SVC_SOCK_ANONYMOUS);
  if (ret < 0) {
   rqstp = ERR_PTR(ret);
@@ -269,7 +269,7 @@ int nfs_callback_up(u32 minorversion, struct rpc_xprt *xprt)
      serv, xprt, &rqstp, &callback_svc);
  if (!minorversion_setup) {
   /* v4.0 callback setup */
-  rqstp = nfs4_callback_up(serv);
+  rqstp = nfs4_callback_up(serv, xprt);
   callback_svc = nfs4_callback_svc;
  }
```

Subject: [PATCH 6/6] NFS: idmap PipeFS notifier introduced
Posted by Stanislav Kinsbursky on Mon, 28 Nov 2011 13:38:49 GMT
View Forum Message <> Reply to Message

This patch subscribes NFS clients to RPC pipefs notifications. Idmap notifier
is registering on NFS module load. This notifier callback is responsible for
creation/destruction of PipeFS idmap pipe dentry for NFS4 clients.

Since ipdmap pipe is created in rpc client pipefs directory, we have make sure,
that this directory has been created already. IOW RPC client notifier callback
has been called already. To achive this, PipeFS notifier priorities has been
introduced (RPC clients notifier priority is greater than NFS idmap one).
But this approach gives another problem: unlink for RPC client directory will
be called before NFS idmap pipe unlink on UMOUNT event and will fail, because
directory is not empty.
The solution, introduced in this patch, is to try to remove client directory
once again after idmap pipe was unlinked. This looks like ugly hack, so
probably it should be replaced in some more elegant way.

Note that no locking required in notifier callback because PipeFS superblock
pointer is passed as an argument from it's creation or destruction routine and
thus we can be sure about it's validity.

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

```
 fs/nfs/client.c                 |    4 +-
```

```
 fs/nfs/idmap.c                  |   75 ++++++++++++++++++++++++++++++++++++++++
 fs/nfs/internal.h               |    4 ++
 include/linux/nfs_idmap.h       |   13 +-----
 include/linux/sunrpc/rpc_pipe_fs.h |    7 +++
 net/sunrpc/clnt.c               |    1
 net/sunrpc/rpc_pipe.c           |   16 ++++++++
 7 files changed, 107 insertions(+), 13 deletions(-)

diff --git a/fs/nfs/client.c b/fs/nfs/client.c
index 58eebb5..4398587 100644
--- a/fs/nfs/client.c
+++ b/fs/nfs/client.c
@@ -52,8 +52,8 @@

 #define NFSDBG_FACILITY  NFSDBG_CLIENT

-static DEFINE_SPINLOCK(nfs_client_lock);
-static LIST_HEAD(nfs_client_list);
+DEFINE_SPINLOCK(nfs_client_lock);
+LIST_HEAD(nfs_client_list);
 static LIST_HEAD(nfs_volume_list);
 static DECLARE_WAIT_QUEUE_HEAD(nfs_client_active_wq);
 #ifdef CONFIG_NFS_V4
diff --git a/fs/nfs/idmap.c b/fs/nfs/idmap.c
index 0680f63..d8fed37 100644
--- a/fs/nfs/idmap.c
+++ b/fs/nfs/idmap.c
@@ -294,6 +294,7 @@ int nfs_map_gid_to_group(const struct nfs_server *server, __u32 gid,
char *buf,
 #include <linux/nfs_fs.h>

 #include "nfs4_fs.h"
+#include "internal.h"

 #define IDMAP_HASH_SZ          128

@@ -449,6 +450,80 @@ nfs_idmap_delete(struct nfs_client *clp)
  kfree(idmap);
 }

+static int __rpc_pipefs_event(struct nfs_client *clp, unsigned long event,
+        struct super_block *sb)
+{
+ int err = 0;
+
+ switch (event) {
+ case RPC_PIPEFS_MOUNT:
+  BUG_ON(clp->cl_rpcclient->cl_dentry == NULL);
```

```
+  err = __nfs_idmap_register(clp->cl_rpcclient->cl_dentry,
+      clp->cl_idmap,
+      clp->cl_idmap->idmap_pipe);
+  break;
+ case RPC_PIPEFS_UMOUNT:
+  if (clp->cl_idmap->idmap_pipe) {
+   struct dentry *parent;
+
+   parent = clp->cl_idmap->idmap_pipe->dentry->d_parent;
+   __nfs_idmap_unregister(clp->cl_idmap->idmap_pipe);
+   /*
+    * Note: This is a dirty hack. SUNRPC hook has been
+    * called already but simple_rmdir() call for the
+    * directory returned with error because of idmap pipe
+    * inside. Thus now we have to remove this directory
+    * here.
+    */
+   if (rpc_rmdir(parent))
+    printk(KERN_ERR "%s: failed to remove clnt dir!\n", __func__);
+  }
+  break;
+ default:
+  printk(KERN_ERR "%s: unknown event: %ld\n", __func__, event);
+  return -ENOTSUPP;
+ }
+ return err;
+}
+
+static int rpc_pipefs_event(struct notifier_block *nb, unsigned long event,
+       void *ptr)
+{
+ struct super_block *sb = ptr;
+ struct nfs_client *clp;
+ int error = 0;
+
+ spin_lock(&nfs_client_lock);
+ list_for_each_entry(clp, &nfs_client_list, cl_share_link) {
+  if (clp->net != sb->s_fs_info)
+   continue;
+  if (clp->rpc_ops != &nfs_v4_clientops)
+   continue;
+  error = __rpc_pipefs_event(clp, event, sb);
+  if (error)
+   break;
+ }
+ spin_unlock(&nfs_client_lock);
+ return error;
+}
```

```
+
+#define PIPEFS_NFS_PRIO  1
+
+static struct notifier_block nfs_idmap_block = {
+ .notifier_call = rpc_pipefs_event,
+ .priority = SUNRPC_PIPEFS_NFS_PRIO,
+};
+
+int nfs_idmap_init(void)
+{
+ return rpc_pipefs_notifier_register(&nfs_idmap_block);
+}
+
+void nfs_idmap_quit(void)
+{
+ rpc_pipefs_notifier_unregister(&nfs_idmap_block);
+}
+
 /*
  * Helper routines for manipulating the hashtable
  */
diff --git a/fs/nfs/internal.h b/fs/nfs/internal.h
index 451acb5..00e8fa5 100644
--- a/fs/nfs/internal.h
+++ b/fs/nfs/internal.h
@@ -182,6 +182,10 @@ static inline void nfs_fs_proc_exit(void)
 {
 }
 #endif
+#ifdef CONFIG_NFS_V4
+extern spinlock_t nfs_client_lock;
+extern struct list_head nfs_client_list;
+#endif

 /* nfs4namespace.c */
 #ifdef CONFIG_NFS_V4
diff --git a/include/linux/nfs_idmap.h b/include/linux/nfs_idmap.h
index ae7d6a3..238c814 100644
--- a/include/linux/nfs_idmap.h
+++ b/include/linux/nfs_idmap.h
@@ -67,11 +67,11 @@ struct idmap_msg {
 struct nfs_client;
 struct nfs_server;

-#ifdef CONFIG_NFS_USE_NEW_IDMAPPER
-
 int nfs_idmap_init(void);
 void nfs_idmap_quit(void);
```

```
+#ifdef CONFIG_NFS_USE_NEW_IDMAPPER
+
 static inline int nfs_idmap_new(struct nfs_client *clp)
 {
  return 0;
@@ -83,15 +83,6 @@ static inline void nfs_idmap_delete(struct nfs_client *clp)

 #else /* CONFIG_NFS_USE_NEW_IDMAPPER not set */

-static inline int nfs_idmap_init(void)
-{
- return 0;
-}
-
-static inline void nfs_idmap_quit(void)
-{
-}
-
 int nfs_idmap_new(struct nfs_client *);
 void nfs_idmap_delete(struct nfs_client *);

diff --git a/include/linux/sunrpc/rpc_pipe_fs.h b/include/linux/sunrpc/rpc_pipe_fs.h
index 0808ed2..7eb0160 100644
--- a/include/linux/sunrpc/rpc_pipe_fs.h
+++ b/include/linux/sunrpc/rpc_pipe_fs.h
@@ -49,6 +49,11 @@ RPC_I(struct inode *inode)
  return container_of(inode, struct rpc_inode, vfs_inode);
 }

+enum {
+ SUNRPC_PIPEFS_NFS_PRIO,
+ SUNRPC_PIPEFS_RPC_PRIO,
+};
+
 extern int rpc_pipefs_notifier_register(struct notifier_block *);
 extern void rpc_pipefs_notifier_unregister(struct notifier_block *);

@@ -78,6 +83,8 @@ extern struct dentry *rpc_create_cache_dir(struct dentry *,
        struct cache_detail *);
 extern void rpc_remove_cache_dir(struct dentry *);

+extern int rpc_rmdir(struct dentry *dentry);
+
 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 *,
diff --git a/net/sunrpc/clnt.c b/net/sunrpc/clnt.c
```

```
index c7237fa..e4f210d 100644
--- a/net/sunrpc/clnt.c
+++ b/net/sunrpc/clnt.c
@@ -234,6 +234,7 @@ static int rpc_pipefs_event(struct notifier_block *nb, unsigned long event,

 static struct notifier_block rpc_clients_block = {
  .notifier_call = rpc_pipefs_event,
+ .priority = SUNRPC_PIPEFS_RPC_PRIO,
 };

 int rpc_clients_notifier_register(void)
diff --git a/net/sunrpc/rpc_pipe.c b/net/sunrpc/rpc_pipe.c
index b80d7bd..e194e32 100644
--- a/net/sunrpc/rpc_pipe.c
+++ b/net/sunrpc/rpc_pipe.c
@@ -597,6 +597,22 @@ static int __rpc_rmdir(struct inode *dir, struct dentry *dentry)
  return ret;
 }

+int rpc_rmdir(struct dentry *dentry)
+{
+ struct dentry *parent;
+ struct inode *dir;
+ int error;
+
+ parent = dget_parent(dentry);
+ dir = parent->d_inode;
+ mutex_lock_nested(&dir->i_mutex, I_MUTEX_PARENT);
+ error = __rpc_rmdir(dir, dentry);
+ mutex_unlock(&dir->i_mutex);
+ dput(parent);
+ return error;
+}
+EXPORT_SYMBOL_GPL(rpc_rmdir);
+
 static int __rpc_unlink(struct inode *dir, struct dentry *dentry)
 {
  int ret;
```

Subject: Re: [PATCH 6/6] NFS: idmap PipeFS notifier introduced
Posted by Myklebust, Trond on Fri, 30 Dec 2011 22:54:03 GMT
View Forum Message <> Reply to Message

On Mon, 2011-11-28 at 17:34 +0300, Stanislav Kinsbursky wrote:
> This patch subscribes NFS clients to RPC pipefs notifications. Idmap notifier
> is registering on NFS module load. This notifier callback is responsible for
> creation/destruction of PipeFS idmap pipe dentry for NFS4 clients.

>
> Since ipdmap pipe is created in rpc client pipefs directory, we have make sure,
> that this directory has been created already. IOW RPC client notifier callback
> has been called already. To achive this, PipeFS notifier priorities has been
> introduced (RPC clients notifier priority is greater than NFS idmap one).
> But this approach gives another problem: unlink for RPC client directory will
> be called before NFS idmap pipe unlink on UMOUNT event and will fail, because
> directory is not empty.
> The solution, introduced in this patch, is to try to remove client directory
> once again after idmap pipe was unlinked. This looks like ugly hack, so
> probably it should be replaced in some more elegant way.
>
> Note that no locking required in notifier callback because PipeFS superblock
> pointer is passed as an argument from it's creation or destruction routine and
> thus we can be sure about it's validity.
>
> Signed-off-by: Stanislav Kinsbursky <skinsbursky@parallels.com>

This patch gives me:

```
Kernel: arch/x86/boot/bzImage is ready  (#3)
  Building modules, stage 2.
  MODPOST 921 modules
ERROR: "nfs_idmap_init" [fs/nfs/nfs.ko] undefined!
ERROR: "nfs_idmap_quit" [fs/nfs/nfs.ko] undefined!
make[2]: *** [__modpost] Error 1
make[1]: *** [modules] Error 2
make: *** [sub-make] Error 2
```

if CONFIG_NFS_V4 is not defined.

Cheers
  Trond

--
Trond Myklebust
Linux NFS client maintainer

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

Subject: Re: [PATCH 6/6] NFS: idmap PipeFS notifier introduced
Posted by Stanislav Kinsbursky on Tue, 10 Jan 2012 10:27:17 GMT
View Forum Message <> Reply to Message

> On Mon, 2011-11-28 at 17:34 +0300, Stanislav Kinsbursky wrote:
>> This patch subscribes NFS clients to RPC pipefs notifications. Idmap notifier
>> is registering on NFS module load. This notifier callback is responsible for
>> creation/destruction of PipeFS idmap pipe dentry for NFS4 clients.
>>
>> Since ipdmap pipe is created in rpc client pipefs directory, we have make sure,
>> that this directory has been created already. IOW RPC client notifier callback
>> has been called already. To achive this, PipeFS notifier priorities has been
>> introduced (RPC clients notifier priority is greater than NFS idmap one).
>> But this approach gives another problem: unlink for RPC client directory will
>> be called before NFS idmap pipe unlink on UMOUNT event and will fail, because
>> directory is not empty.
>> The solution, introduced in this patch, is to try to remove client directory
>> once again after idmap pipe was unlinked. This looks like ugly hack, so
>> probably it should be replaced in some more elegant way.
>>
>> Note that no locking required in notifier callback because PipeFS superblock
>> pointer is passed as an argument from it's creation or destruction routine and
>> thus we can be sure about it's validity.
>>
>> Signed-off-by: Stanislav Kinsbursky<skinsbursky@parallels.com>
>
> This patch gives me:
>
> Kernel: arch/x86/boot/bzImage is ready (#3)
>   Building modules, stage 2.
>   MODPOST 921 modules
> ERROR: "nfs_idmap_init" [fs/nfs/nfs.ko] undefined!
> ERROR: "nfs_idmap_quit" [fs/nfs/nfs.ko] undefined!
> make[2]: *** [__modpost] Error 1
> make[1]: *** [modules] Error 2
> make: *** [sub-make] Error 2
>
> if CONFIG_NFS_V4 is not defined.
>

Sorry.
Will resend soon.

> Cheers
>   Trond
>

--
Best regards,
Stanislav Kinsbursky