Subject: [PATCH 00/11] SUNRPC: make sysctl per network namespcase context
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:45:50 GMT
View Forum Message <> Reply to Message

This patch set make SUNRPC sysctl created per network namspace context.
It provides the following functionality:
1) Debug sysctl's ({rpc,nfs,nlm,nfsd}_debug) content is modifyable only from init_net network context.
2) Write to rpc_debug prints active RPC tasks in sysctl's creator networks context.
3) xs tunables are per-net now and modified through per-net sysctl's. IOW, sysctl creator context is used to get tunable during sysctl read/write access (not current tone).

The following series consists of:

Stanislav Kinsbursky (11):
SYSCTL: export root ans set handling routines
SUNRPC: use syctl path instead of dummy parent table
SUNRPC: sysctl root for debug table introduced
SUNRPC: per-net sysctl's set introduced
SUNRPC: register debug sysctl table per network namespace
SUNRPC: register xs_tunables sysctl table per network namespace
SUNRPC: xs tunables per network namespace introduced
SUNRPC: use per-net xs tunables instead of static ones
SUNRPC: remove xs_tcp_fin_timeout variable
SUNRPC: allow debug flags modifications only from init_net
SUNRPC: sysctl table for rpc_debug introduced

```
include/linux/sunrpc/debug.h    |    9 ++
include/linux/sunrpc/sched.h    |    1
include/linux/sunrpc/xprtsock.h |    3 +
include/linux/sysctl.h          |    1
kernel/sysctl.c                 |   11 +++
net/sunrpc/netns.h              |   12 +++
net/sunrpc/sunrpc_syms.c        |   27 ++++++-
net/sunrpc/sysctl.c             |  160 +++++++++++++++++++++++++++++++++++-------
net/sunrpc/xprtrdma/transport.c |   11 ---
net/sunrpc/xprtsock.c           |  147 ++++++++++++++++++++++++++----------
10 files changed, 293 insertions(+), 89 deletions(-)
```

Subject: [PATCH 11/11] SUNRPC: sysctl table for rpc_debug introduced
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:15 GMT

This patch provides showing of pending RPC tasks for right namespace in case of
write oparation to "rpc_debug" sysctl.

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

```
---
 include/linux/sunrpc/sched.h |    1
 net/sunrpc/netns.h           |    1
 net/sunrpc/sysctl.c          |   87 ++++++++++++++++++++++++++++++++++++++-----
 3 files changed, 77 insertions(+), 12 deletions(-)

diff --git a/include/linux/sunrpc/sched.h b/include/linux/sunrpc/sched.h
index b16243a..02f5fce 100644
--- a/include/linux/sunrpc/sched.h
+++ b/include/linux/sunrpc/sched.h
@@ -244,7 +244,6 @@ int  rpciod_up(void);
 void  rpciod_down(void);
 int  __rpc_wait_for_completion_task(struct rpc_task *task, int (*)(void *));
 #ifdef RPC_DEBUG
-struct net;
 void  rpc_show_tasks(struct net *);
 #endif
 int  rpc_init_mempool(void);
diff --git a/net/sunrpc/netns.h b/net/sunrpc/netns.h
index 23d110d..0926705 100644
--- a/net/sunrpc/netns.h
+++ b/net/sunrpc/netns.h
@@ -16,6 +16,7 @@ struct sunrpc_net {
  struct ctl_table_set sysctls;
  struct ctl_table_header *debug_ctl_header;
  struct ctl_table_header *xs_tunables_header;
+ struct ctl_table_header *rpc_debug_ctl_header;
 #endif
  unsigned int xprt_udp_slot_table_entries;
  unsigned int xprt_tcp_slot_table_entries;
diff --git a/net/sunrpc/sysctl.c b/net/sunrpc/sysctl.c
index 224b075..52f6fb5 100644
--- a/net/sunrpc/sysctl.c
+++ b/net/sunrpc/sysctl.c
@@ -44,6 +44,15 @@ EXPORT_SYMBOL_GPL(nlm_debug);

 static ctl_table  debug_table[];

+static ctl_table rpc_debug_table[] = {
+ {
+  .procname = "rpc_debug",
+  .maxlen  = sizeof(int),
```

```
+  .mode  = 0644,
+ },
+ { }
+};
+
 struct ctl_path sunrpc_path[] = {
  { .procname = "sunrpc", },
  { },
@@ -73,6 +82,48 @@ struct ctl_table_header *register_sunrpc_sysctl(struct net *net,
 }
 EXPORT_SYMBOL_GPL(register_sunrpc_sysctl);

+static int proc_rpcdebug(ctl_table *table, int write,
+    void __user *buffer, size_t *lenp, loff_t *ppos);
+
+static int register_rpc_debug_table(struct net *net)
+{
+ struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+ struct ctl_table *table;
+
+ table = rpc_debug_table;
+ if (!net_eq(net, &init_net)) {
+  table = kmemdup(rpc_debug_table, sizeof(rpc_debug_table),
+    GFP_KERNEL);
+  if (table == NULL)
+   goto err_alloc;
+ }
+ table[0].data = net;
+ table[0].proc_handler = proc_rpcdebug;
+
+ sn->rpc_debug_ctl_header = register_sunrpc_sysctl(net, table);
+ if (sn->rpc_debug_ctl_header == NULL)
+  goto err_reg;
+ return 0;
+
+err_reg:
+ if (!net_eq(net, &init_net))
+  kfree(table);
+err_alloc:
+ return -ENOMEM;
+}
+
+static void unregister_rpc_debug_table(struct net *net)
+{
+ struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+ struct ctl_table *table;
+
+ table = sn->rpc_debug_ctl_header->ctl_table_arg;
```

```
+ unregister_sysctl_table(sn->rpc_debug_ctl_header);
+ sn->rpc_debug_ctl_header = NULL;
+ if (!net_eq(net, &init_net))
+  kfree(table);
+}
+
 int debug_sysctl_init(struct net *net)
 {
  struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
@@ -80,14 +131,23 @@ int debug_sysctl_init(struct net *net)
  setup_sysctl_set(&sn->sysctls, NULL, NULL);
  sn->debug_ctl_header = register_sunrpc_sysctl(net, debug_table);
  if (sn->debug_ctl_header == NULL)
-  return -ENOMEM;
+  goto err_debug;
+ if (register_rpc_debug_table(net) < 0)
+  goto err_rpc_debug;
  return 0;
+
+err_rpc_debug:
+ unregister_sysctl_table(sn->debug_ctl_header);
+ sn->debug_ctl_header = NULL;
+err_debug:
+ return -ENOMEM;
 }

 void debug_sysctl_exit(struct net *net)
 {
  struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);

+ unregister_rpc_debug_table(net);
  unregister_sysctl_table(sn->debug_ctl_header);
  sn->debug_ctl_header = NULL;
  WARN_ON(!list_empty(&sn->sysctls.list));
@@ -158,9 +218,6 @@ proc_dodebug(ctl_table *table, int write,
   left--, s++;
   if (net_eq(current->nsproxy->net_ns, &init_net))
   *(unsigned int *) table->data = value;
-  /* Display the RPC tasks on writing to rpc_debug */
-  if (strcmp(table->procname, "rpc_debug") == 0)
-   rpc_show_tasks(&init_net);
  } else {
   if (!access_ok(VERIFY_WRITE, buffer, left))
   return -EFAULT;
@@ -183,15 +240,23 @@ done:
 }
```

```
+static int
+proc_rpcdebug(ctl_table *table, int write,
+    void __user *buffer, size_t *lenp, loff_t *ppos)
+{
+ int err;
+ struct ctl_table tmp = *table;
+
+ tmp.data = &rpc_debug;
+ err = proc_dodebug(&tmp, write, buffer, lenp, ppos);
+ if (!err && write)
+  /* Display the RPC tasks on writing to rpc_debug */
+  rpc_show_tasks(table->data);
+ return err;
+}
+
 static ctl_table debug_table[] = {
  {
-  .procname = "rpc_debug",
-  .data  = &rpc_debug,
-  .maxlen  = sizeof(int),
-  .mode  = 0644,
-  .proc_handler = proc_dodebug
- },
- {
   .procname = "nfs_debug",
   .data  = &nfs_debug,
   .maxlen  = sizeof(int),
```

Subject: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:33 GMT
View Forum Message <> Reply to Message

These routines are required for making SUNRPC sysctl's per network namespace context.

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

```
---
 include/linux/sysctl.h |    1 +
 kernel/sysctl.c        |   11 +++++++++++
 2 files changed, 12 insertions(+), 0 deletions(-)

diff --git a/include/linux/sysctl.h b/include/linux/sysctl.h
index 703cfa3..be586a9 100644
--- a/include/linux/sysctl.h
+++ b/include/linux/sysctl.h
@@ -1084,6 +1084,7 @@ struct ctl_path {
```

```
 };

 void register_sysctl_root(struct ctl_table_root *root);
+void unregister_sysctl_root(struct ctl_table_root *root);
 struct ctl_table_header *__register_sysctl_paths(
  struct ctl_table_root *root, struct nsproxy *namespaces,
  const struct ctl_path *path, struct ctl_table *table);
diff --git a/kernel/sysctl.c b/kernel/sysctl.c
index ae27196..fb016a9 100644
--- a/kernel/sysctl.c
+++ b/kernel/sysctl.c
@@ -1700,6 +1700,15 @@ void register_sysctl_root(struct ctl_table_root *root)
  list_add_tail(&root->root_list, &sysctl_table_root.root_list);
  spin_unlock(&sysctl_lock);
 }
+EXPORT_SYMBOL_GPL(register_sysctl_root);
+
+void unregister_sysctl_root(struct ctl_table_root *root)
+{
+ spin_lock(&sysctl_lock);
+ list_del(&root->root_list);
+ spin_unlock(&sysctl_lock);
+}
+EXPORT_SYMBOL_GPL(unregister_sysctl_root);

 /*
  * sysctl_perm does NOT grant the superuser all rights automatically, because
@@ -1925,6 +1934,7 @@ struct ctl_table_header *__register_sysctl_paths(

  return header;
 }
+EXPORT_SYMBOL_GPL(__register_sysctl_paths);

 /**
  * register_sysctl_table_path - register a sysctl table hierarchy
@@ -2007,6 +2017,7 @@ void setup_sysctl_set(struct ctl_table_set *p,
  p->parent = parent ? parent : &sysctl_table_root.default_set;
  p->is_seen = is_seen;
 }
+EXPORT_SYMBOL_GPL(setup_sysctl_set);

 #else /* !CONFIG_SYSCTL */
 struct ctl_table_header *register_sysctl_table(struct ctl_table * table)
```

Subject: [PATCH 02/11] SUNRPC: use syctl path instead of dummy parent table
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:35 GMT

This is a cleanup patch. Parent sunrpc table is redundant. Syctl path can be used instead.

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

```
---
 include/linux/sunrpc/debug.h    |    4 ++++
 net/sunrpc/sysctl.c             |   25 +++++++++++++++-----------
 net/sunrpc/xprtrdma/transport.c |   11 +----------
 net/sunrpc/xprtsock.c           |   11 +----------
 4 files changed, 20 insertions(+), 31 deletions(-)

diff --git a/include/linux/sunrpc/debug.h b/include/linux/sunrpc/debug.h
index c2786f2..b5e0c46 100644
--- a/include/linux/sunrpc/debug.h
+++ b/include/linux/sunrpc/debug.h
@@ -63,8 +63,12 @@ extern unsigned int  nlm_debug;
  * Sysctl interface for RPC debugging
  */
 #ifdef RPC_DEBUG
+
 void  rpc_register_sysctl(void);
 void  rpc_unregister_sysctl(void);
+
+struct ctl_table_header *register_sunrpc_sysctl(struct ctl_table *);
+
 #endif

 #endif /* __KERNEL__ */
diff --git a/net/sunrpc/sysctl.c b/net/sunrpc/sysctl.c
index af7d339..64c0034 100644
--- a/net/sunrpc/sysctl.c
+++ b/net/sunrpc/sysctl.c
@@ -40,13 +40,25 @@ EXPORT_SYMBOL_GPL(nlm_debug);
 #ifdef RPC_DEBUG

 static struct ctl_table_header *sunrpc_table_header;
-static ctl_table  sunrpc_table[];
+static ctl_table  debug_table[];
+
+struct ctl_path sunrpc_path[] = {
+ { .procname = "sunrpc", },
+ { },
+};
+
+struct ctl_table_header *register_sunrpc_sysctl(struct ctl_table *table)
+{
```

```
+ return register_sysctl_paths(sunrpc_path, table);
+
+}
+EXPORT_SYMBOL_GPL(register_sunrpc_sysctl);

 void
 rpc_register_sysctl(void)
 {
  if (!sunrpc_table_header)
-  sunrpc_table_header = register_sysctl_table(sunrpc_table);
+  sunrpc_table_header = register_sunrpc_sysctl(debug_table);
 }

 void
@@ -173,13 +185,4 @@ static ctl_table debug_table[] = {
  { }
 };

-static ctl_table sunrpc_table[] = {
- {
-  .procname = "sunrpc",
-  .mode  = 0555,
-  .child  = debug_table
- },
- { }
-};
-
 #endif
diff --git a/net/sunrpc/xprtrdma/transport.c b/net/sunrpc/xprtrdma/transport.c
index b446e10..53a8622 100644
--- a/net/sunrpc/xprtrdma/transport.c
+++ b/net/sunrpc/xprtrdma/transport.c
@@ -137,15 +137,6 @@ static ctl_table xr_tunables_table[] = {
  { },
 };

-static ctl_table sunrpc_table[] = {
- {
-  .procname = "sunrpc",
-  .mode  = 0555,
-  .child  = xr_tunables_table
- },
- { },
-};
-
 #endif

 static struct rpc_xprt_ops xprt_rdma_procs; /* forward reference */
```

```
@@ -771,7 +762,7 @@ static int __init xprt_rdma_init(void)

 #ifdef RPC_DEBUG
  if (!sunrpc_table_header)
-  sunrpc_table_header = register_sysctl_table(sunrpc_table);
+  sunrpc_table_header = register_sunrpc_sysctl(xr_tunables_table);
 #endif
  return 0;
 }
diff --git a/net/sunrpc/xprtsock.c b/net/sunrpc/xprtsock.c
index 55472c4..6f1be96 100644
--- a/net/sunrpc/xprtsock.c
+++ b/net/sunrpc/xprtsock.c
@@ -142,15 +142,6 @@ static ctl_table xs_tunables_table[] = {
  { },
 };

-static ctl_table sunrpc_table[] = {
- {
-  .procname = "sunrpc",
-  .mode  = 0555,
-  .child  = xs_tunables_table
- },
- { },
-};
-
 #endif

 /*
@@ -2887,7 +2878,7 @@ int init_socket_xprt(void)
 {
 #ifdef RPC_DEBUG
  if (!sunrpc_table_header)
-  sunrpc_table_header = register_sysctl_table(sunrpc_table);
+  sunrpc_table_header = register_sunrpc_sysctl(xs_tunables_table);
 #endif

  xprt_register_transport(&xs_local_transport);
```

Subject: [PATCH 03/11] SUNRPC: sysctl root for debug table introduced
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:37 GMT
View Forum Message <> Reply to Message

Sysctl root is required for handliong sysctl dentries per network namespace
context. IOW, it's "lookup" method will be used to find per-net sysctl's set in
further patches.
Also this patch modifies sysctl registering helpers to make them use new sysctl

root.

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

```
---
 net/sunrpc/sysctl.c |   16 +++++++++++++---
 1 files changed, 13 insertions(+), 3 deletions(-)

diff --git a/net/sunrpc/sysctl.c b/net/sunrpc/sysctl.c
index 64c0034..f6e7da2 100644
--- a/net/sunrpc/sysctl.c
+++ b/net/sunrpc/sysctl.c
@@ -39,6 +39,9 @@ EXPORT_SYMBOL_GPL(nlm_debug);

 #ifdef RPC_DEBUG

+#include <linux/nsproxy.h>
+#include <net/net_namespace.h>
+
 static struct ctl_table_header *sunrpc_table_header;
 static ctl_table  debug_table[];

@@ -47,18 +50,24 @@ struct ctl_path sunrpc_path[] = {
  { },
 };

+static struct ctl_table_root sunrpc_debug_root = {
+};
+
 struct ctl_table_header *register_sunrpc_sysctl(struct ctl_table *table)
 {
- return register_sysctl_paths(sunrpc_path, table);
-
+ return __register_sysctl_paths(&sunrpc_debug_root, current->nsproxy,
+           sunrpc_path, table);
 }
 EXPORT_SYMBOL_GPL(register_sunrpc_sysctl);

 void
 rpc_register_sysctl(void)
 {
- if (!sunrpc_table_header)
+ if (!sunrpc_table_header) {
+  setup_sysctl_set(&sunrpc_debug_root.default_set, NULL, NULL);
+  register_sysctl_root(&sunrpc_debug_root);
   sunrpc_table_header = register_sunrpc_sysctl(debug_table);
+ }
 }
```

```
 void
@@ -66,6 +75,7 @@ rpc_unregister_sysctl(void)
 {
  if (sunrpc_table_header) {
   unregister_sysctl_table(sunrpc_table_header);
+  unregister_sysctl_root(&sunrpc_debug_root);
   sunrpc_table_header = NULL;
  }
 }
```

Subject: [PATCH 05/11] SUNRPC: register debug sysctl table per network namespace
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:39 GMT
View Forum Message <> Reply to Message

This patch introduces per-net sysctl header for debug table and moves debug table registration to per-net opreations.
Also, this patch moves SUNRPC sysctl root registration prior to per-net operations, since now they depends on it.

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

```
---
 net/sunrpc/netns.h       |    1 +
 net/sunrpc/sunrpc_syms.c |    9 ++++++---
 net/sunrpc/sysctl.c      |   17 +++++++----------
 3 files changed, 14 insertions(+), 13 deletions(-)

diff --git a/net/sunrpc/netns.h b/net/sunrpc/netns.h
index a949ddb..05ce3b8 100644
--- a/net/sunrpc/netns.h
+++ b/net/sunrpc/netns.h
@@ -14,6 +14,7 @@ struct sunrpc_net {
  spinlock_t rpc_client_lock;
 #ifdef RPC_DEBUG
  struct ctl_table_set sysctls;
+ struct ctl_table_header *debug_ctl_header;
 #endif
 };

diff --git a/net/sunrpc/sunrpc_syms.c b/net/sunrpc/sunrpc_syms.c
index 44759e9..0fd156a 100644
--- a/net/sunrpc/sunrpc_syms.c
+++ b/net/sunrpc/sunrpc_syms.c
@@ -90,18 +90,21 @@ init_sunrpc(void)
```

```
  cache_initialize();

- err = register_pernet_subsys(&sunrpc_net_ops);
- if (err)
-  goto out4;
 #ifdef RPC_DEBUG
  rpc_register_sysctl();
 #endif
+ err = register_pernet_subsys(&sunrpc_net_ops);
+ if (err)
+  goto out4;
  cache_register(&unix_gid_cache);
  svc_init_xprt_sock(); /* svc sock transport */
  init_socket_xprt(); /* clnt sock transport */
  return 0;

 out4:
+#ifdef RPC_DEBUG
+ rpc_unregister_sysctl();
+#endif
  rpcauth_remove_module();
 out3:
  rpc_destroy_mempool();
diff --git a/net/sunrpc/sysctl.c b/net/sunrpc/sysctl.c
index adebf0a..eda80cf 100644
--- a/net/sunrpc/sysctl.c
+++ b/net/sunrpc/sysctl.c
@@ -42,7 +42,6 @@ EXPORT_SYMBOL_GPL(nlm_debug);
 #include <linux/nsproxy.h>
 #include "netns.h"

-static struct ctl_table_header *sunrpc_table_header;
 static ctl_table  debug_table[];

 struct ctl_path sunrpc_path[] = {
@@ -79,6 +78,9 @@ int debug_sysctl_init(struct net *net)
  struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);

  setup_sysctl_set(&sn->sysctls, NULL, NULL);
+ sn->debug_ctl_header = register_sunrpc_sysctl(net, debug_table);
+ if (sn->debug_ctl_header == NULL)
+  return -ENOMEM;
  return 0;
 }

@@ -86,26 +88,21 @@ void debug_sysctl_exit(struct net *net)
 {
  struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
```

```
+ unregister_sysctl_table(sn->debug_ctl_header);
+ sn->debug_ctl_header = NULL;
  WARN_ON(!list_empty(&sn->sysctls.list));
 }

 void
 rpc_register_sysctl(void)
 {
- if (!sunrpc_table_header) {
-  register_sysctl_root(&sunrpc_debug_root);
-  sunrpc_table_header = register_sunrpc_sysctl(&init_net, debug_table);
- }
+ register_sysctl_root(&sunrpc_debug_root);
 }

 void
 rpc_unregister_sysctl(void)
 {
- if (sunrpc_table_header) {
-  unregister_sysctl_table(sunrpc_table_header);
-  unregister_sysctl_root(&sunrpc_debug_root);
-  sunrpc_table_header = NULL;
- }
+ unregister_sysctl_root(&sunrpc_debug_root);
 }

 static int proc_do_xprt(ctl_table *table, int write,
```

Subject: [PATCH 04/11] SUNRPC: per-net sysctl's set introduced
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:39 GMT
View Forum Message <> Reply to Message

This patch introduces per-net sysctl set and it's initialization routines. Also
it modifies sysctl's registering helpers to make them use these sets.

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

```
---
 include/linux/sunrpc/debug.h    |    7 ++++++-
 net/sunrpc/netns.h              |    3 +++
 net/sunrpc/sunrpc_syms.c        |   13 +++++++++++-
 net/sunrpc/sysctl.c             |   38 ++++++++++++++++++++++++++++++++++-----
 net/sunrpc/xprtrdma/transport.c |    2 +-
 net/sunrpc/xprtsock.c           |    2 +-
 6 files changed, 56 insertions(+), 9 deletions(-)
```

```
diff --git a/include/linux/sunrpc/debug.h b/include/linux/sunrpc/debug.h
index b5e0c46..f2d825c 100644
--- a/include/linux/sunrpc/debug.h
+++ b/include/linux/sunrpc/debug.h
@@ -67,7 +67,12 @@ extern unsigned int  nlm_debug;
 void  rpc_register_sysctl(void);
 void  rpc_unregister_sysctl(void);

-struct ctl_table_header *register_sunrpc_sysctl(struct ctl_table *);
+struct net;
+struct ctl_table;
+int debug_sysctl_init(struct net *);
+void debug_sysctl_exit(struct net *);
+struct ctl_table_header *register_sunrpc_sysctl(struct net *,
+      struct ctl_table *);

 #endif

diff --git a/net/sunrpc/netns.h b/net/sunrpc/netns.h
index 6010c46..a949ddb 100644
--- a/net/sunrpc/netns.h
+++ b/net/sunrpc/netns.h
@@ -12,6 +12,9 @@ struct sunrpc_net {

  struct list_head all_clients;
  spinlock_t rpc_client_lock;
+#ifdef RPC_DEBUG
+ struct ctl_table_set sysctls;
+#endif
 };

 extern int sunrpc_net_id;
diff --git a/net/sunrpc/sunrpc_syms.c b/net/sunrpc/sunrpc_syms.c
index e57aa10..44759e9 100644
--- a/net/sunrpc/sunrpc_syms.c
+++ b/net/sunrpc/sunrpc_syms.c
@@ -38,11 +38,19 @@ static __net_init int sunrpc_init_net(struct net *net)
  err = ip_map_cache_create(net);
  if (err)
   goto err_ipmap;
-
+#ifdef RPC_DEBUG
+ err = debug_sysctl_init(net);
+ if (err)
+  goto err_sysctl;
+#endif
  INIT_LIST_HEAD(&sn->all_clients);
  spin_lock_init(&sn->rpc_client_lock);
```

```
  return 0;

+#ifdef RPC_DEBUG
+err_sysctl:
+ ip_map_cache_destroy(net);
+#endif
 err_ipmap:
  rpc_proc_exit(net);
 err_proc:
@@ -51,6 +59,9 @@ err_proc:

 static __net_exit void sunrpc_exit_net(struct net *net)
 {
+#ifdef RPC_DEBUG
+ debug_sysctl_exit(net);
+#endif
  ip_map_cache_destroy(net);
  rpc_proc_exit(net);
 }
diff --git a/net/sunrpc/sysctl.c b/net/sunrpc/sysctl.c
index f6e7da2..adebf0a 100644
--- a/net/sunrpc/sysctl.c
+++ b/net/sunrpc/sysctl.c
@@ -40,7 +40,7 @@ EXPORT_SYMBOL_GPL(nlm_debug);
 #ifdef RPC_DEBUG

 #include <linux/nsproxy.h>
-#include <net/net_namespace.h>
+#include "netns.h"

 static struct ctl_table_header *sunrpc_table_header;
 static ctl_table  debug_table[];
@@ -50,23 +50,51 @@ struct ctl_path sunrpc_path[] = {
  { },
 };

+static struct ctl_table_set *
+sunrpc_debug_lookup(struct ctl_table_root *root, struct nsproxy *namespaces)
+{
+ struct sunrpc_net *sn = net_generic(namespaces->net_ns, sunrpc_net_id);
+
+ return &sn->sysctls;
+}
+
 static struct ctl_table_root sunrpc_debug_root = {
+ .lookup = sunrpc_debug_lookup,
 };
```

```
-struct ctl_table_header *register_sunrpc_sysctl(struct ctl_table *table)
+struct ctl_table_header *register_sunrpc_sysctl(struct net *net,
+       struct ctl_table *table)
 {
- return __register_sysctl_paths(&sunrpc_debug_root, current->nsproxy,
+ struct nsproxy namespaces;
+
+ namespaces = *current->nsproxy;
+ namespaces.net_ns = net;
+ return __register_sysctl_paths(&sunrpc_debug_root, &namespaces,
            sunrpc_path, table);
 }
 EXPORT_SYMBOL_GPL(register_sunrpc_sysctl);

+int debug_sysctl_init(struct net *net)
+{
+ struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+
+ setup_sysctl_set(&sn->sysctls, NULL, NULL);
+ return 0;
+}
+
+void debug_sysctl_exit(struct net *net)
+{
+ struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+
+ WARN_ON(!list_empty(&sn->sysctls.list));
+}
+
 void
 rpc_register_sysctl(void)
 {
  if (!sunrpc_table_header) {
-  setup_sysctl_set(&sunrpc_debug_root.default_set, NULL, NULL);
   register_sysctl_root(&sunrpc_debug_root);
-  sunrpc_table_header = register_sunrpc_sysctl(debug_table);
+  sunrpc_table_header = register_sunrpc_sysctl(&init_net, debug_table);
  }
 }

diff --git a/net/sunrpc/xprtrdma/transport.c b/net/sunrpc/xprtrdma/transport.c
index 53a8622..4a63b35 100644
--- a/net/sunrpc/xprtrdma/transport.c
+++ b/net/sunrpc/xprtrdma/transport.c
@@ -762,7 +762,7 @@ static int __init xprt_rdma_init(void)

 #ifdef RPC_DEBUG
  if (!sunrpc_table_header)
```

```
-  sunrpc_table_header = register_sunrpc_sysctl(xr_tunables_table);
+  sunrpc_table_header = register_sunrpc_sysctl(&init_net, xr_tunables_table);
 #endif
  return 0;
 }
diff --git a/net/sunrpc/xprtsock.c b/net/sunrpc/xprtsock.c
index 6f1be96..1acc9a7 100644
--- a/net/sunrpc/xprtsock.c
+++ b/net/sunrpc/xprtsock.c
@@ -2878,7 +2878,7 @@ int init_socket_xprt(void)
 {
 #ifdef RPC_DEBUG
  if (!sunrpc_table_header)
-  sunrpc_table_header = register_sunrpc_sysctl(xs_tunables_table);
+  sunrpc_table_header = register_sunrpc_sysctl(&init_net, xs_tunables_table);
 #endif

  xprt_register_transport(&xs_local_transport);
```

Subject: [PATCH 06/11] SUNRPC: register xs_tunables sysctl table per network namespace

Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:41 GMT

View Forum Message <> Reply to Message

This patch introduces per-net sysctl header for xs_tunables table and moves
xs_tunables table registration to per-net opreations.

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

```
---
 include/linux/sunrpc/xprtsock.h |    3 ++
 net/sunrpc/netns.h              |    1 +
 net/sunrpc/sunrpc_syms.c        |    7 +++++
 net/sunrpc/xprtsock.c           |   52 ++++++++++++++++++++++++++++++----------
 4 files changed, 49 insertions(+), 14 deletions(-)

diff --git a/include/linux/sunrpc/xprtsock.h b/include/linux/sunrpc/xprtsock.h
index 3f14a02..3ba4920 100644
--- a/include/linux/sunrpc/xprtsock.h
+++ b/include/linux/sunrpc/xprtsock.h
@@ -9,6 +9,9 @@

 #ifdef __KERNEL__

+int socket_sysctl_init(struct net *);
+void socket_sysctl_exit(struct net *);
+
```

```
 int  init_socket_xprt(void);
 void  cleanup_socket_xprt(void);

diff --git a/net/sunrpc/netns.h b/net/sunrpc/netns.h
index 05ce3b8..da825e5 100644
--- a/net/sunrpc/netns.h
+++ b/net/sunrpc/netns.h
@@ -15,6 +15,7 @@ struct sunrpc_net {
 #ifdef RPC_DEBUG
  struct ctl_table_set sysctls;
  struct ctl_table_header *debug_ctl_header;
+ struct ctl_table_header *xs_tunables_header;
 #endif
 };

diff --git a/net/sunrpc/sunrpc_syms.c b/net/sunrpc/sunrpc_syms.c
index 0fd156a..2b76e18 100644
--- a/net/sunrpc/sunrpc_syms.c
+++ b/net/sunrpc/sunrpc_syms.c
@@ -42,12 +42,18 @@ static __net_init int sunrpc_init_net(struct net *net)
  err = debug_sysctl_init(net);
  if (err)
   goto err_sysctl;
+
+ err = socket_sysctl_init(net);
+ if (err)
+  goto err_sock_ctl;
 #endif
  INIT_LIST_HEAD(&sn->all_clients);
  spin_lock_init(&sn->rpc_client_lock);
  return 0;

 #ifdef RPC_DEBUG
+err_sock_ctl:
+ debug_sysctl_exit(net);
 err_sysctl:
  ip_map_cache_destroy(net);
 #endif
@@ -60,6 +66,7 @@ err_proc:
 static __net_exit void sunrpc_exit_net(struct net *net)
 {
 #ifdef RPC_DEBUG
+ socket_sysctl_exit(net);
  debug_sysctl_exit(net);
 #endif
  ip_map_cache_destroy(net);
diff --git a/net/sunrpc/xprtsock.c b/net/sunrpc/xprtsock.c
index 1acc9a7..9e42db6 100644
```

```
--- a/net/sunrpc/xprtsock.c
+++ b/net/sunrpc/xprtsock.c
@@ -74,14 +74,14 @@ static unsigned int xs_tcp_fin_timeout __read_mostly =
XS_TCP_LINGER_TO;

 #ifdef RPC_DEBUG

+#include "netns.h"
+
 static unsigned int min_slot_table_size = RPC_MIN_SLOT_TABLE;
 static unsigned int max_slot_table_size = RPC_MAX_SLOT_TABLE;
 static unsigned int max_tcp_slot_table_limit = RPC_MAX_SLOT_TABLE_LIMIT;
 static unsigned int xprt_min_resvport_limit = RPC_MIN_RESVPORT;
 static unsigned int xprt_max_resvport_limit = RPC_MAX_RESVPORT;

-static struct ctl_table_header *sunrpc_table_header;
-
 /*
  * FIXME: changing the UDP slot table size should also resize the UDP
  *        socket buffers for existing UDP transports
@@ -2870,17 +2870,48 @@ static struct xprt_class xs_bc_tcp_transport = {
  .setup  = xs_setup_bc_tcp,
 };

+#ifdef RPC_DEBUG
+int create_xs_tunables_table(struct net *net)
+{
+ struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+
+ sn->xs_tunables_header = register_sunrpc_sysctl(net, xs_tunables_table);
+ if (sn->xs_tunables_header == NULL)
+  return -ENOMEM;
+ return 0;
+
+}
+void destroy_xs_tunables_table(struct net *net)
+{
+ struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+ unregister_sysctl_table(sn->xs_tunables_header);
+ sn->xs_tunables_header = NULL;
+}
+#endif
+
+
+int socket_sysctl_init(struct net *net)
+{
+#ifdef RPC_DEBUG
+ return create_xs_tunables_table(net);
```

```
+#else
+ return 0;
+#endif
+}
+
+void socket_sysctl_exit(struct net *net)
+{
+#ifdef RPC_DEBUG
+ destroy_xs_tunables_table(net);
+#endif
+}
+
 /**
  * init_socket_xprt - set up xprtsock's sysctls, register with RPC client
  *
  */
 int init_socket_xprt(void)
 {
-#ifdef RPC_DEBUG
- if (!sunrpc_table_header)
-  sunrpc_table_header = register_sunrpc_sysctl(&init_net, xs_tunables_table);
-#endif
-
  xprt_register_transport(&xs_local_transport);
  xprt_register_transport(&xs_udp_transport);
  xprt_register_transport(&xs_tcp_transport);
@@ -2895,13 +2926,6 @@ int init_socket_xprt(void)
  */
 void cleanup_socket_xprt(void)
 {
-#ifdef RPC_DEBUG
- if (sunrpc_table_header) {
-  unregister_sysctl_table(sunrpc_table_header);
-  sunrpc_table_header = NULL;
- }
-#endif
-
  xprt_unregister_transport(&xs_local_transport);
  xprt_unregister_transport(&xs_udp_transport);
  xprt_unregister_transport(&xs_tcp_transport);
```

Subject: [PATCH 07/11] SUNRPC: xs tunables per network namespace introduced
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:43 GMT
View Forum Message <> Reply to Message

This patch introduces independent sets of xs tunables for every network
namespace context. Their value is accessible via per-net sysctls.

Static tunables left as a storage for module params and now used for
initialization of new network namespace tunables set.

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

```
---
 net/sunrpc/netns.h    |    6 ++++++
 net/sunrpc/xprtsock.c |   47 ++++++++++++++++++++++++++++++++++++++++---------
 2 files changed, 44 insertions(+), 9 deletions(-)

diff --git a/net/sunrpc/netns.h b/net/sunrpc/netns.h
index da825e5..23d110d 100644
--- a/net/sunrpc/netns.h
+++ b/net/sunrpc/netns.h
@@ -17,6 +17,12 @@ struct sunrpc_net {
  struct ctl_table_header *debug_ctl_header;
  struct ctl_table_header *xs_tunables_header;
 #endif
+ unsigned int xprt_udp_slot_table_entries;
+ unsigned int xprt_tcp_slot_table_entries;
+ unsigned int xprt_max_tcp_slot_table_entries;
+ unsigned int xprt_min_resvport;
+ unsigned int xprt_max_resvport;
+ unsigned int xs_tcp_fin_timeout;
 };

 extern int sunrpc_net_id;
diff --git a/net/sunrpc/xprtsock.c b/net/sunrpc/xprtsock.c
index 9e42db6..fe4b04b 100644
--- a/net/sunrpc/xprtsock.c
+++ b/net/sunrpc/xprtsock.c
@@ -89,7 +89,6 @@ static unsigned int xprt_max_resvport_limit = RPC_MAX_RESVPORT;
 static ctl_table xs_tunables_table[] = {
  {
   .procname = "udp_slot_table_entries",
-  .data  = &xprt_udp_slot_table_entries,
   .maxlen  = sizeof(unsigned int),
   .mode  = 0644,
   .proc_handler = proc_dointvec_minmax,
@@ -98,7 +97,6 @@ static ctl_table xs_tunables_table[] = {
  },
  {
   .procname = "tcp_slot_table_entries",
-  .data  = &xprt_tcp_slot_table_entries,
   .maxlen  = sizeof(unsigned int),
   .mode  = 0644,
   .proc_handler = proc_dointvec_minmax,
@@ -107,7 +105,6 @@ static ctl_table xs_tunables_table[] = {
```

```
 },
 {
  .procname = "tcp_max_slot_table_entries",
-  .data  = &xprt_max_tcp_slot_table_entries,
  .maxlen  = sizeof(unsigned int),
  .mode  = 0644,
  .proc_handler = proc_dointvec_minmax,
@@ -116,7 +113,6 @@ static ctl_table xs_tunables_table[] = {
 },
 {
  .procname = "min_resvport",
-  .data  = &xprt_min_resvport,
  .maxlen  = sizeof(unsigned int),
  .mode  = 0644,
  .proc_handler = proc_dointvec_minmax,
@@ -125,7 +121,6 @@ static ctl_table xs_tunables_table[] = {
 },
 {
  .procname = "max_resvport",
-  .data  = &xprt_max_resvport,
  .maxlen  = sizeof(unsigned int),
  .mode  = 0644,
  .proc_handler = proc_dointvec_minmax,
@@ -134,7 +129,6 @@ static ctl_table xs_tunables_table[] = {
 },
 {
  .procname = "tcp_fin_timeout",
-  .data  = &xs_tcp_fin_timeout,
  .maxlen  = sizeof(xs_tcp_fin_timeout),
  .mode  = 0644,
  .proc_handler = proc_dointvec_jiffies,
@@ -2874,24 +2868,59 @@ static struct xprt_class xs_bc_tcp_transport = {
 int create_xs_tunables_table(struct net *net)
 {
  struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+ struct ctl_table *table;
+
+ table = xs_tunables_table;
+ if (!net_eq(net, &init_net)) {
+  table = kmemdup(xs_tunables_table, sizeof(xs_tunables_table),
+    GFP_KERNEL);
+  if (table == NULL)
+   goto err_alloc;
+ }
+
+ table[0].data = &sn->xprt_udp_slot_table_entries;
+ table[1].data = &sn->xprt_tcp_slot_table_entries;
+ table[2].data = &sn->xprt_max_tcp_slot_table_entries;
```

```
+ table[3].data = &sn->xprt_min_resvport;
+ table[4].data = &sn->xprt_max_resvport;
+ table[5].data = &sn->xs_tcp_fin_timeout;

- sn->xs_tunables_header = register_sunrpc_sysctl(net, xs_tunables_table);
+ sn->xs_tunables_header = register_sunrpc_sysctl(net, table);
  if (sn->xs_tunables_header == NULL)
-  return -ENOMEM;
+  goto err_reg;
+
  return 0;

+err_reg:
+ if (!net_eq(net, &init_net))
+  kfree(table);
+err_alloc:
+ return -ENOMEM;
 }
+
 void destroy_xs_tunables_table(struct net *net)
 {
  struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+ struct ctl_table *table;
+
+ table = sn->xs_tunables_header->ctl_table_arg;
  unregister_sysctl_table(sn->xs_tunables_header);
  sn->xs_tunables_header = NULL;
+ if (!net_eq(net, &init_net))
+  kfree(table);
 }
 #endif

-
 int socket_sysctl_init(struct net *net)
 {
+ struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+
+ sn->xprt_udp_slot_table_entries = xprt_udp_slot_table_entries;
+ sn->xprt_tcp_slot_table_entries = xprt_tcp_slot_table_entries;
+ sn->xprt_max_tcp_slot_table_entries = xprt_max_tcp_slot_table_entries;
+ sn->xprt_min_resvport = xprt_min_resvport;
+ sn->xprt_max_resvport = xprt_max_resvport;
+ sn->xs_tcp_fin_timeout = xs_tcp_fin_timeout;
 #ifdef RPC_DEBUG
  return create_xs_tunables_table(net);
 #else
```

Subject: [PATCH 09/11] SUNRPC: remove xs_tcp_fin_timeout variable
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:43 GMT
View Forum Message <> Reply to Message

This is a cleanup patch. Static xs_tcp_fin_timeout is not needed anymore.

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

```
---
 net/sunrpc/xprtsock.c |    5 ++---
 1 files changed, 2 insertions(+), 3 deletions(-)

diff --git a/net/sunrpc/xprtsock.c b/net/sunrpc/xprtsock.c
index 49219d2..610a74a 100644
--- a/net/sunrpc/xprtsock.c
+++ b/net/sunrpc/xprtsock.c
@@ -61,7 +61,6 @@ unsigned int xprt_min_resvport = RPC_DEF_MIN_RESVPORT;
 unsigned int xprt_max_resvport = RPC_DEF_MAX_RESVPORT;

 #define XS_TCP_LINGER_TO (15U * HZ)
-static unsigned int xs_tcp_fin_timeout __read_mostly = XS_TCP_LINGER_TO;

 /*
  * We can register our own files under /proc/sys/sunrpc by
@@ -129,7 +128,7 @@ static ctl_table xs_tunables_table[] = {
  },
  {
   .procname = "tcp_fin_timeout",
-  .maxlen  = sizeof(xs_tcp_fin_timeout),
+  .maxlen  = sizeof(unsigned int),
   .mode  = 0644,
   .proc_handler = proc_dointvec_jiffies,
  },
@@ -2930,7 +2929,7 @@ int socket_sysctl_init(struct net *net)
  sn->xprt_max_tcp_slot_table_entries = xprt_max_tcp_slot_table_entries;
  sn->xprt_min_resvport = xprt_min_resvport;
  sn->xprt_max_resvport = xprt_max_resvport;
- sn->xs_tcp_fin_timeout = xs_tcp_fin_timeout;
+ sn->xs_tcp_fin_timeout = XS_TCP_LINGER_TO;
 #ifdef RPC_DEBUG
  return create_xs_tunables_table(net);
 #else
```

Subject: [PATCH 10/11] SUNRPC: allow debug flags modifications only from init_net
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:45 GMT
View Forum Message <> Reply to Message

Debug flags are global (i.e. fo all namespaces). So probably, it is better to restrict write access and allow it only to processes with "init_net" network namespace.

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

```
---
 net/sunrpc/sysctl.c |    3 ++-
 1 files changed, 2 insertions(+), 1 deletions(-)

diff --git a/net/sunrpc/sysctl.c b/net/sunrpc/sysctl.c
index eda80cf..224b075 100644
--- a/net/sunrpc/sysctl.c
+++ b/net/sunrpc/sysctl.c
@@ -156,7 +156,8 @@ proc_dodebug(ctl_table *table, int write,
    return -EINVAL;
   while (left && isspace(*s))
    left--, s++;
-  *(unsigned int *) table->data = value;
+  if (net_eq(current->nsproxy->net_ns, &init_net))
+   *(unsigned int *) table->data = value;
   /* Display the RPC tasks on writing to rpc_debug */
   if (strcmp(table->procname, "rpc_debug") == 0)
    rpc_show_tasks(&init_net);
```

Subject: [PATCH 08/11] SUNRPC: use per-net xs tunables instead of static ones
Posted by Stanislav Kinsbursky on Wed, 14 Dec 2011 10:47:51 GMT
View Forum Message <> Reply to Message

This patch replaces references to static tunables with per-net ones.

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

```
---
 net/sunrpc/xprtsock.c |   42 ++++++++++++++++++++++++++++----------------
 1 files changed, 26 insertions(+), 16 deletions(-)

diff --git a/net/sunrpc/xprtsock.c b/net/sunrpc/xprtsock.c
index fe4b04b..49219d2 100644
--- a/net/sunrpc/xprtsock.c
+++ b/net/sunrpc/xprtsock.c
@@ -1458,6 +1458,7 @@ static void xs_sock_mark_closed(struct rpc_xprt *xprt)
 static void xs_tcp_state_change(struct sock *sk)
 {
  struct rpc_xprt *xprt;
+ struct sunrpc_net *sn = net_generic(sk->sk_net, sunrpc_net_id);
```

```
 read_lock_bh(&sk->sk_callback_lock);
 if (!(xprt = xprt_from_sock(sk)))
@@ -1496,7 +1497,7 @@ static void xs_tcp_state_change(struct sock *sk)
  clear_bit(XPRT_CONNECTED, &xprt->state);
  clear_bit(XPRT_CLOSE_WAIT, &xprt->state);
  smp_mb__after_clear_bit();
-  xs_tcp_schedule_linger_timeout(xprt, xs_tcp_fin_timeout);
+  xs_tcp_schedule_linger_timeout(xprt, sn->xs_tcp_fin_timeout);
  break;
 case TCP_CLOSE_WAIT:
  /* The server initiated a shutdown of the socket */
@@ -1512,7 +1513,7 @@ static void xs_tcp_state_change(struct sock *sk)
  break;
 case TCP_LAST_ACK:
  set_bit(XPRT_CLOSING, &xprt->state);
-  xs_tcp_schedule_linger_timeout(xprt, xs_tcp_fin_timeout);
+  xs_tcp_schedule_linger_timeout(xprt, sn->xs_tcp_fin_timeout);
  smp_mb__before_clear_bit();
  clear_bit(XPRT_CONNECTED, &xprt->state);
  smp_mb__after_clear_bit();
@@ -1652,11 +1653,13 @@ static void xs_udp_timer(struct rpc_task *task)
 xprt_adjust_cwnd(task, -ETIMEDOUT);
}

-static unsigned short xs_get_random_port(void)
+static unsigned short xs_get_random_port(struct net *net)
{
- unsigned short range = xprt_max_resvport - xprt_min_resvport;
+ struct sunrpc_net *sn = net_generic(net, sunrpc_net_id);
+
+ unsigned short range = sn->xprt_max_resvport - sn->xprt_min_resvport;
 unsigned short rand = (unsigned short) net_random() % range;
- return rand + xprt_min_resvport;
+ return rand + sn->xprt_min_resvport;
}

/**
@@ -1678,18 +1681,21 @@ static unsigned short xs_get_srcport(struct sock_xprt *transport)
 unsigned short port = transport->srcport;

 if (port == 0 && transport->xprt.resvport)
-  port = xs_get_random_port();
+  port = xs_get_random_port(transport->xprt.xprt_net);
 return port;
}

static unsigned short xs_next_srcport(struct sock_xprt *transport, unsigned short port)
{
```

```
+ struct sunrpc_net *sn = net_generic(transport->xprt.xprt_net,
+         sunrpc_net_id);
+
  if (transport->srcport != 0)
   transport->srcport = 0;
  if (!transport->xprt.resvport)
   return 0;
- if (port <= xprt_min_resvport || port > xprt_max_resvport)
-  return xprt_max_resvport;
+ if (port <= sn->xprt_min_resvport || port > sn->xprt_max_resvport)
+  return sn->xprt_max_resvport;
  return --port;
 }
 static int xs_bind(struct sock_xprt *transport, struct socket *sock)
@@ -2541,9 +2547,10 @@ static struct rpc_xprt *xs_setup_local(struct xprt_create *args)
  struct sock_xprt *transport;
  struct rpc_xprt *xprt;
  struct rpc_xprt *ret;
+ struct sunrpc_net *sn = net_generic(args->net, sunrpc_net_id);

- xprt = xs_setup_xprt(args, xprt_tcp_slot_table_entries,
-   xprt_max_tcp_slot_table_entries);
+ xprt = xs_setup_xprt(args, sn->xprt_tcp_slot_table_entries,
+   sn->xprt_max_tcp_slot_table_entries);
  if (IS_ERR(xprt))
   return xprt;
  transport = container_of(xprt, struct sock_xprt, xprt);
@@ -2606,9 +2613,10 @@ static struct rpc_xprt *xs_setup_udp(struct xprt_create *args)
  struct rpc_xprt *xprt;
  struct sock_xprt *transport;
  struct rpc_xprt *ret;
+ struct sunrpc_net *sn = net_generic(args->net, sunrpc_net_id);

- xprt = xs_setup_xprt(args, xprt_udp_slot_table_entries,
-   xprt_udp_slot_table_entries);
+ xprt = xs_setup_xprt(args, sn->xprt_udp_slot_table_entries,
+   sn->xprt_udp_slot_table_entries);
  if (IS_ERR(xprt))
   return xprt;
  transport = container_of(xprt, struct sock_xprt, xprt);
@@ -2683,9 +2691,10 @@ static struct rpc_xprt *xs_setup_tcp(struct xprt_create *args)
  struct rpc_xprt *xprt;
  struct sock_xprt *transport;
  struct rpc_xprt *ret;
+ struct sunrpc_net *sn = net_generic(args->net, sunrpc_net_id);

- xprt = xs_setup_xprt(args, xprt_tcp_slot_table_entries,
-   xprt_max_tcp_slot_table_entries);
```

```
+ xprt = xs_setup_xprt(args, sn->xprt_tcp_slot_table_entries,
+   sn->xprt_max_tcp_slot_table_entries);
  if (IS_ERR(xprt))
   return xprt;
  transport = container_of(xprt, struct sock_xprt, xprt);
@@ -2754,6 +2763,7 @@ static struct rpc_xprt *xs_setup_bc_tcp(struct xprt_create *args)
  struct sock_xprt *transport;
  struct svc_sock *bc_sock;
  struct rpc_xprt *ret;
+ struct sunrpc_net *sn = net_generic(args->net, sunrpc_net_id);

  if (args->bc_xprt->xpt_bc_xprt) {
   /*
@@ -2764,8 +2774,8 @@ static struct rpc_xprt *xs_setup_bc_tcp(struct xprt_create *args)
    */
   return args->bc_xprt->xpt_bc_xprt;
  }
- xprt = xs_setup_xprt(args, xprt_tcp_slot_table_entries,
-   xprt_tcp_slot_table_entries);
+ xprt = xs_setup_xprt(args, sn->xprt_tcp_slot_table_entries,
+   sn->xprt_tcp_slot_table_entries);
  if (IS_ERR(xprt))
   return xprt;
  transport = container_of(xprt, struct sock_xprt, xprt);
```

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by ebiederm on Sat, 17 Dec 2011 22:25:14 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> These routines are required for making SUNRPC sysctl's per network namespace
> context.

Why does sunrpc require it's own sysctl root? You should be able to use
the generic per network namespace root and call it good.

What makes register_net_sysctl_table and register_net_sysctl_ro_table
unsuitable for sunrpc. I skimmed through your patches and I haven't
seen anything obvious.

Eric

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

```
>  include/linux/sysctl.h |    1 +
>  kernel/sysctl.c        |   11 +++++++++++
>  2 files changed, 12 insertions(+), 0 deletions(-)
>
> diff --git a/include/linux/sysctl.h b/include/linux/sysctl.h
> index 703cfa3..be586a9 100644
> --- a/include/linux/sysctl.h
> +++ b/include/linux/sysctl.h
> @@ -1084,6 +1084,7 @@ struct ctl_path {
>  };
>
>  void register_sysctl_root(struct ctl_table_root *root);
> +void unregister_sysctl_root(struct ctl_table_root *root);
>  struct ctl_table_header *__register_sysctl_paths(
>   struct ctl_table_root *root, struct nsproxy *namespaces,
>   const struct ctl_path *path, struct ctl_table *table);
> diff --git a/kernel/sysctl.c b/kernel/sysctl.c
> index ae27196..fb016a9 100644
> --- a/kernel/sysctl.c
> +++ b/kernel/sysctl.c
> @@ -1700,6 +1700,15 @@ void register_sysctl_root(struct ctl_table_root *root)
>   list_add_tail(&root->root_list, &sysctl_table_root.root_list);
>   spin_unlock(&sysctl_lock);
>  }
> +EXPORT_SYMBOL_GPL(register_sysctl_root);
> +
> +void unregister_sysctl_root(struct ctl_table_root *root)
> +{
> + spin_lock(&sysctl_lock);
> + list_del(&root->root_list);
> + spin_unlock(&sysctl_lock);
> +}
> +EXPORT_SYMBOL_GPL(unregister_sysctl_root);
>
>  /*
>   * sysctl_perm does NOT grant the superuser all rights automatically, because
> @@ -1925,6 +1934,7 @@ struct ctl_table_header *__register_sysctl_paths(
>
>   return header;
>  }
> +EXPORT_SYMBOL_GPL(__register_sysctl_paths);
>
>  /**
>   * register_sysctl_table_path - register a sysctl table hierarchy
> @@ -2007,6 +2017,7 @@ void setup_sysctl_set(struct ctl_table_set *p,
>   p->parent = parent ? parent : &sysctl_table_root.default_set;
>   p->is_seen = is_seen;
>  }
```

> +EXPORT_SYMBOL_GPL(setup_sysctl_set);
>
> #else /* !CONFIG_SYSCTL */
> struct ctl_table_header *register_sysctl_table(struct ctl_table * table)
>
> --
> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html
> Please read the FAQ at http://www.tux.org/lkml/

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by Stanislav Kinsbursky on Mon, 19 Dec 2011 08:56:52 GMT
View Forum Message <> Reply to Message

18.12.2011 02:25, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>
>> These routines are required for making SUNRPC sysctl's per network namespace
>> context.
>
> Why does sunrpc require it's own sysctl root? You should be able to use
> the generic per network namespace root and call it good.
>
> What makes register_net_sysctl_table and register_net_sysctl_ro_table
> unsuitable for sunrpc. I skimmed through your patches and I haven't
> seen anything obvious.
>
> Eric
>

Hello, Eric. Sorry for the lack of information.
I was considering two ways how to make these sysctl per net ns:

1) Use register_net_sysctl_table and register_net_sysctl_ro_table as you
mentioned. This was easy and cheap, but also means, than all user-space
programs, tuning SUNRPC will be broken (since all sysctl currently located
in"/proc/sys/sunprc/").

2) Export sysctl root creation routines and make per-net SUNRPC sysctl root.
This approach allows to make any part of sysctl tree per namespace context and
thus leave user-space stuff unchanged.

BTW, NFS and LockD also have it's sysctls ("/proc/sys/fs/nfs/").
And also because of them I've decided, that it would be better to export SYSCTL
root creation routines instead of breaking compatibility for all NFS layers by
moving all sysctl under /proc/sys/net/ directory.

Do you feel that it was a bad decision?

```
>
>> Signed-off-by: Stanislav Kinsbursky<skinsbursky@parallels.com>
>>
>> ---
>>   include/linux/sysctl.h |    1 +
>>   kernel/sysctl.c        |   11 +++++++++++
>>   2 files changed, 12 insertions(+), 0 deletions(-)
>>
>> diff --git a/include/linux/sysctl.h b/include/linux/sysctl.h
>> index 703cfa3..be586a9 100644
>> --- a/include/linux/sysctl.h
>> +++ b/include/linux/sysctl.h
>> @@ -1084,6 +1084,7 @@ struct ctl_path {
>>   };
>>
>>   void register_sysctl_root(struct ctl_table_root *root);
>> +void unregister_sysctl_root(struct ctl_table_root *root);
>>   struct ctl_table_header *__register_sysctl_paths(
>>    struct ctl_table_root *root, struct nsproxy *namespaces,
>>    const struct ctl_path *path, struct ctl_table *table);
>> diff --git a/kernel/sysctl.c b/kernel/sysctl.c
>> index ae27196..fb016a9 100644
>> --- a/kernel/sysctl.c
>> +++ b/kernel/sysctl.c
>> @@ -1700,6 +1700,15 @@ void register_sysctl_root(struct ctl_table_root *root)
>>    list_add_tail(&root->root_list,&sysctl_table_root.root_list);
>>    spin_unlock(&sysctl_lock);
>>   }
>> +EXPORT_SYMBOL_GPL(register_sysctl_root);
>> +
>> +void unregister_sysctl_root(struct ctl_table_root *root)
>> +{
>> + spin_lock(&sysctl_lock);
>> + list_del(&root->root_list);
>> + spin_unlock(&sysctl_lock);
>> +}
>> +EXPORT_SYMBOL_GPL(unregister_sysctl_root);
>>
>>   /*
>>    * sysctl_perm does NOT grant the superuser all rights automatically, because
>> @@ -1925,6 +1934,7 @@ struct ctl_table_header *__register_sysctl_paths(
>>
>>    return header;
>>   }
>> +EXPORT_SYMBOL_GPL(__register_sysctl_paths);
```

>>
>> /**
>>   * register_sysctl_table_path - register a sysctl table hierarchy
>> @@ -2007,6 +2017,7 @@ void setup_sysctl_set(struct ctl_table_set *p,
>>   p->parent = parent ? parent :&sysctl_table_root.default_set;
>>   p->is_seen = is_seen;
>> }
>> +EXPORT_SYMBOL_GPL(setup_sysctl_set);
>>
>> #else /* !CONFIG_SYSCTL */
>> struct ctl_table_header *register_sysctl_table(struct ctl_table * table)
>>
>> --
>> To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
>> the body of a message to majordomo@vger.kernel.org
>> More majordomo info at http://vger.kernel.org/majordomo-info.html
>> Please read the FAQ at http://www.tux.org/lkml/
> --
> To unsubscribe from this list: send the line "unsubscribe linux-nfs" in
> the body of a message to majordomo@vger.kernel.org
> More majordomo info at http://vger.kernel.org/majordomo-info.html

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by ebiederm on Mon, 19 Dec 2011 10:14:30 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> 18.12.2011 02:25, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>
>>> These routines are required for making SUNRPC sysctl's per network namespace
>>> context.
>>
>> Why does sunrpc require it's own sysctl root? You should be able to use
>> the generic per network namespace root and call it good.
>>
>> What makes register_net_sysctl_table and register_net_sysctl_ro_table
>> unsuitable for sunrpc. I skimmed through your patches and I haven't
>> seen anything obvious.
>>
>> Eric

>>
>
> Hello, Eric. Sorry for the lack of information.
> I was considering two ways how to make these sysctl per net ns:
>
> 1) Use register_net_sysctl_table and register_net_sysctl_ro_table as you
> mentioned. This was easy and cheap, but also means, than all user-space
> programs, tuning SUNRPC will be broken (since all sysctl currently located
> in"/proc/sys/sunprc/").

Nope. That is a misunderstanding. register_net_sysctl_table works for
anything under /proc/sys.

> 2) Export sysctl root creation routines and make per-net SUNRPC sysctl
> root. This approach allows to make any part of sysctl tree per namespace context
> and thus leave user-space stuff unchanged.
>
> BTW, NFS and LockD also have it's sysctls ("/proc/sys/fs/nfs/").
> And also because of them I've decided, that it would be better to export SYSCTL
> root creation routines instead of breaking compatibility for all NFS layers by
> moving all sysctl under /proc/sys/net/ directory.
>
> Do you feel that it was a bad decision?

I think it was a misinformed decision.

I fully support not breaking userspace by moving where the sysctls files
are. If something sounds like I am suggesting moving sysctl files there
is a miscommunication somewhere.

The concept of a sysctl root as I had envisioned it and essentially as it
is implemented was a per namespace sysctl tree. Those sysctl trees are
then unioned together when presented to user space. There should only
be one root per namespace.

In practice what this means is that register_net_sysctl_table should
work for any sysctl file anywhere under /proc/sys. I think
register_net_sysctl_table is the right solution for your problem. The
only possible caveat I can think of is you might hit Al's performance
optimizations and need to create a common empty directory first with
register_sysctl_paths.

....
That said since I am in the process of rewriting things some of this
may change a little bit, but hopefully not in ways that immediately
effect the users of register_sysctl_table.

Don't use register_net_sysctl_ro_table. I think what the implementors actually wanted was register_net_sysctl_table(&init_net, ...) and didn't know it.

Don't put subdirectories in your sysctl tables. Use a ctl_path to specify the entire directory where the files should show up. Generally the code is easier to read in that form, and the code is simpler to deal with if we don't have to worry about directories.

Don't play with the sysctl roots. It is my intention to completely kill them off and replace them by moving the per net sysctl tree under /proc/<pid>/sys/. Leaving behind symlinks in /proc/sys/net and I guess ultimately in /proc/sys/sunrpc/ and /proc/sys/fs/nfs... Which actually seems to better describe your mental model.

Thank you for mentioning /proc/sys/fs/nfs. That is a case I hadn't thought about. In thinking about it I see some deficiencies in my rewrite that I need to correct before I push that code.

Eric

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by Stanislav Kinsbursky on Mon, 19 Dec 2011 12:22:50 GMT
View Forum Message <> Reply to Message

19.12.2011 14:15, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>
>> 18.12.2011 02:25, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>>
>>>> These routines are required for making SUNRPC sysctl's per network namespace
>>>> context.
>>>
>>> Why does sunrpc require it's own sysctl root? You should be able to use
>>> the generic per network namespace root and call it good.
>>>
>>> What makes register_net_sysctl_table and register_net_sysctl_ro_table
>>> unsuitable for sunrpc. I skimmed through your patches and I haven't
>>> seen anything obvious.
>>>
>>> Eric
>>>
>>
>> Hello, Eric. Sorry for the lack of information.
>> I was considering two ways how to make these sysctl per net ns:
>>

>> 1) Use register_net_sysctl_table and register_net_sysctl_ro_table as you
>> mentioned. This was easy and cheap, but also means, than all user-space
>> programs, tuning SUNRPC will be broken (since all sysctl currently located
>> in"/proc/sys/sunprc/").
>
> Nope. That is a misunderstanding. register_net_sysctl_table works for
> anything under /proc/sys.
>
>> 2) Export sysctl root creation routines and make per-net SUNRPC sysctl
>> root. This approach allows to make any part of sysctl tree per namespace context
>> and thus leave user-space stuff unchanged.
>>
>> BTW, NFS and LockD also have it's sysctls ("/proc/sys/fs/nfs/").
>> And also because of them I've decided, that it would be better to export SYSCTL
>> root creation routines instead of breaking compatibility for all NFS layers by
>> moving all sysctl under /proc/sys/net/ directory.
>>
>> Do you feel that it was a bad decision?
>
> I think it was a misinformed decision.
>
> I fully support not breaking userspace by moving where the sysctls files
> are. If something sounds like I am suggesting moving sysctl files there
> is a miscommunication somewhere.
>
> The concept of a sysctl root as I had envisioned it and essentially as it
> is implemented was a per namespace sysctl tree. Those sysctl trees are
> then unioned together when presented to user space. There should only
> be one root per namespace.
>
> In practice what this means is that register_net_sysctl_table should
> work for any sysctl file anywhere under /proc/sys. I think
> register_net_sysctl_table is the right solution for your problem. The
> only possible caveat I can think of is you might hit Al's performance
> optimizations and need to create a common empty directory first with
> register_sysctl_paths.
>
>

Sorry, but I forgot to mention one more important goal I would like to achieve:
I want to manage sysctl's variables in context of mount owner, but not viewer one.
IOW imagine, that we have one two network namespaces: "A" and "B". Both of them
have it's own net sysctl's root. And we have per-net sysctl "/proc/sys/var".
And for ns "A" variable was set to 0, and for "B" - to 1.
And B's "/proc/sys/var" is accessible from "A" namespace
("/chroot_path/proc/sys/var" for example).
With this configuration I want to read "1" from both namespaces:

owner "B" (/proc/sys/var) and "A" ("/chroot_path/proc/sys/var").
Looks like simple using of register_net_sysctl_table doesn't allow me this,
because current net ns is used. And to achieve this goal I need my own sysctl
set for SUNRPC like it was done for network namespaces.

> ....
> That said since I am in the process of rewriting things some of this
> may change a little bit, but hopefully not in ways that immediately
> effect the users of register_sysctl_table.
>
> Don't use register_net_sysctl_ro_table. I think what the implementors
> actually wanted was register_net_sysctl_table(&init_net, ...) and didn't
> know it.
>
> Don't put subdirectories in your sysctl tables. Use a ctl_path to
> specify the entire directory where the files should show up. Generally
> the code is easier to read in that form, and the code is simpler to deal
> with if we don't have to worry about directories.
>
> Don't play with the sysctl roots. It is my intention to completely kill
> them off and replace them by moving the per net sysctl tree under
> /proc/<pid>/sys/. Leaving behind symlinks in /proc/sys/net and I guess
> ultimately in /proc/sys/sunrpc/ and /proc/sys/fs/nfs... Which actually
> seems to better describe your mental model.
>

I'm afraid, that this approach this not allow me to achieve the goal, mentioned
above, because current->nsproxy->net_ns will be used during lookup.
Or maybe I misunderstanding here?

> Thank you for mentioning /proc/sys/fs/nfs. That is a case I hadn't
> thought about. In thinking about it I see some deficiencies in my
> rewrite that I need to correct before I push that code.
>

Was glad to be usefull.

> Eric

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by ebiederm on Mon, 19 Dec 2011 16:37:19 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

>> In practice what this means is that register_net_sysctl_table should
>> work for any sysctl file anywhere under /proc/sys. I think
>> register_net_sysctl_table is the right solution for your problem. The
>> only possible caveat I can think of is you might hit Al's performance
>> optimizations and need to create a common empty directory first with
>> register_sysctl_paths.
>
> Sorry, but I forgot to mention one more important goal I would like to achieve:
> I want to manage sysctl's variables in context of mount owner, but not viewer one.
> IOW imagine, that we have one two network namespaces: "A" and "B". Both of them
> have it's own net sysctl's root. And we have per-net sysctl "/proc/sys/var".
> And for ns "A" variable was set to 0, and for "B" - to 1.
> And B's "/proc/sys/var" is accessible from "A" namespace
> ("/chroot_path/proc/sys/var" for example).
> With this configuration I want to read "1" from both namespaces:
> owner "B" (/proc/sys/var) and "A" ("/chroot_path/proc/sys/var").
> Looks like simple using of register_net_sysctl_table doesn't allow me this,
> because current net ns is used. And to achieve this goal I need my own sysctl
> set for SUNRPC like it was done for network namespaces.

Doing that independently of the rest of the sysctls is pretty horrible
and confusing to users. What I am planning might suit your needs and
if not we need to talk some more about how to get the vfs to do
something reasonable.

>> That said since I am in the process of rewriting things some of this
>> may change a little bit, but hopefully not in ways that immediately
>> effect the users of register_sysctl_table.
>>
>> Don't use register_net_sysctl_ro_table. I think what the implementors
>> actually wanted was register_net_sysctl_table(&init_net, ...) and didn't
>> know it.
>>
>> Don't put subdirectories in your sysctl tables. Use a ctl_path to
>> specify the entire directory where the files should show up. Generally
>> the code is easier to read in that form, and the code is simpler to deal
>> with if we don't have to worry about directories.
>>
>> Don't play with the sysctl roots. It is my intention to completely kill
>> them off and replace them by moving the per net sysctl tree under
>> /proc/<pid>/sys/. Leaving behind symlinks in /proc/sys/net and I guess
>> ultimately in /proc/sys/sunrpc/ and /proc/sys/fs/nfs... Which actually
>> seems to better describe your mental model.

>>
>
>
> I'm afraid, that this approach this not allow me to achieve the goal, mentioned
> above, because current->nsproxy->net_ns will be used during lookup.
> Or maybe I misunderstanding here?

What I hope to do is to stop using current, and to behave like
/proc/net. Aka a per process view under /proc/<pid>/sys that matches
the namespaces of the specified process.

The VFS really hates my use of current in the sysctl case, and I intend
to stop.

I need to run and catch my plane. It doesn't look like I will have
access to this email address for the next two weeks :(

Eric

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by Stanislav Kinsbursky on Mon, 19 Dec 2011 17:24:52 GMT
View Forum Message <> Reply to Message

19.12.2011 20:37, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>
> Doing that independently of the rest of the sysctls is pretty horrible
> and confusing to users. What I am planning might suit your needs and
> if not we need to talk some more about how to get the vfs to do
> something reasonable.
>

Ok, Eric. Would be glad to discuss your sysctls plans.
But actually you already know my needs: I would like to make sysctls work in the
way like sysfs does: i.e. content of files depends on mount maker - not viewer.

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by ebiederm on Tue, 03 Jan 2012 03:48:52 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> 19.12.2011 20:37, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>
>> Doing that independently of the rest of the sysctls is pretty horrible
>> and confusing to users. What I am planning might suit your needs and
>> if not we need to talk some more about how to get the vfs to do
>> something reasonable.
>>
>
> Ok, Eric. Would be glad to discuss your sysctls plans.
> But actually you already know my needs: I would like to make sysctls work in the
> way like sysfs does: i.e. content of files depends on mount maker -
> not viewer.

What drives the desire to have sysctls depend on the mount maker?
Especially what drives that desire not to have it have a /proc/<pid>/sys
directory that reflects the sysctls for a given process.

Eric

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by Stanislav Kinsbursky on Tue, 10 Jan 2012 10:38:07 GMT
View Forum Message <> Reply to Message

03.01.2012 07:49, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>
>> 19.12.2011 20:37, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>>
>>> Doing that independently of the rest of the sysctls is pretty horrible
>>> and confusing to users. What I am planning might suit your needs and
>>> if not we need to talk some more about how to get the vfs to do
>>> something reasonable.
>>>
>>
>> Ok, Eric. Would be glad to discuss your sysctls plans.
>> But actually you already know my needs: I would like to make sysctls work in the
>> way like sysfs does: i.e. content of files depends on mount maker -
>> not viewer.
>
> What drives the desire to have sysctls depend on the mount maker?

Because we can (will, actually) have nested fs root's for containers. IOW,
container's root will be accessible from it's creator context. And I want to
tune container's fs from creators context.

> Especially what drives that desire not to have it have a /proc/<pid>/sys
> directory that reflects the sysctls for a given process.
>

This is not so important for me, where to access sysctl's. But I'm worrying
about backward compatibility. IOW, I'm afraid of changing path
"/proc/sys/sunprc/*" to "/proc/<pid>/sys/sunrpc". This would break a lot of
user-space programs.

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by ebiederm on Tue, 10 Jan 2012 22:38:17 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> 03.01.2012 07:49, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>
>>> 19.12.2011 20:37, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>>>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>>>
>>>> Doing that independently of the rest of the sysctls is pretty horrible
>>>> and confusing to users. What I am planning might suit your needs and
>>>> if not we need to talk some more about how to get the vfs to do
>>>> something reasonable.
>>>>
>>>
>>> Ok, Eric. Would be glad to discuss your sysctls plans.
>>> But actually you already know my needs: I would like to make sysctls work in the
>>> way like sysfs does: i.e. content of files depends on mount maker -
>>> not viewer.
>>
>> What drives the desire to have sysctls depend on the mount maker?
>
> Because we can (will, actually) have nested fs root's for containers. IOW,
> container's root will be accessible from it's creator context. And I want to
> tune container's fs from creators context.

Tuning the child context from the parent context is an entirely
reasonable thing to do. To affect a namespace that is not yours
the requirement is simply that we don't use current to lookup the
sysctl. So what I am proposing should work for your case.

>> Especially what drives that desire not to have it have a /proc/<pid>/sys
>> directory that reflects the sysctls for a given process.
>>
>
> This is not so important for me, where to access sysctl's. But I'm worrying
> about backward compatibility. IOW, I'm afraid of changing path
> "/proc/sys/sunprc/*" to "/proc/<pid>/sys/sunrpc". This would break a lot of
> user-space programs.

The part that keeps it all working is by adding a symlink from /proc/sys
to /proc/self/sys. That technique has worked well for /proc/net, and I
don't expect there will be any problems with /proc/sys either. It is
possible but is very rare for the introduction of a symlink in a path
to cause problems.

Eric

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by Stanislav Kinsbursky on Wed, 11 Jan 2012 09:47:10 GMT
View Forum Message <> Reply to Message

11.01.2012 02:39, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>
>> 03.01.2012 07:49, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>>
>>>> 19.12.2011 20:37, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>>>>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>>>>
>>>>> Doing that independently of the rest of the sysctls is pretty horrible
>>>>> and confusing to users. What I am planning might suit your needs and
>>>>> if not we need to talk some more about how to get the vfs to do
>>>>> something reasonable.
>>>>>
>>>>
>>>> Ok, Eric. Would be glad to discuss your sysctls plans.
>>>> But actually you already know my needs: I would like to make sysctls work in the
>>>> way like sysfs does: i.e. content of files depends on mount maker -
>>>> not viewer.
>>>
>>> What drives the desire to have sysctls depend on the mount maker?
>>
>> Because we can (will, actually) have nested fs root's for containers. IOW,
>> container's root will be accessible from it's creator context. And I want to
>> tune container's fs from creators context.

>
> Tuning the child context from the parent context is an entirely
> reasonable thing to do. To affect a namespace that is not yours
> the requirement is simply that we don't use current to lookup the
> sysctl. So what I am proposing should work for your case.
>

Could you explain, what are you proposing?
I still don't know any details about it.

>>> Especially what drives that desire not to have it have a /proc/<pid>/sys
>>> directory that reflects the sysctls for a given process.
>>>
>>
>> This is not so important for me, where to access sysctl's. But I'm worrying
>> about backward compatibility. IOW, I'm afraid of changing path
>> "/proc/sys/sunprc/*" to "/proc/<pid>/sys/sunrpc". This would break a lot of
>> user-space programs.
>
> The part that keeps it all working is by adding a symlink from /proc/sys
> to /proc/self/sys. That technique has worked well for /proc/net, and I
> don't expect there will be any problems with /proc/sys either. It is
> possible but is very rare for the introduction of a symlink in a path
> to cause problems.
>

Probably I don't understand you, but as I see it now, symlink to "/proc/self/"
is unacceptable because of the following:
1) will be used current context (any) instead of desired one
1) if CT has other pid namespace - then we just have broken link.

> Eric
>

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by ebiederm on Wed, 11 Jan 2012 17:20:17 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> 11.01.2012 02:39, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:

>>
>>> 03.01.2012 07:49, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>>>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>>>
>>>>> 19.12.2011 20:37, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>>>>>> Stanislav Kinsbursky<skinsbursky@parallels.com> writes:
>>>>>>
>>>>>> Doing that independently of the rest of the sysctls is pretty horrible
>>>>>> and confusing to users. What I am planning might suit your needs and
>>>>>> if not we need to talk some more about how to get the vfs to do
>>>>>> something reasonable.
>>>>>>
>>>>>
>>>>> Ok, Eric. Would be glad to discuss your sysctls plans.
>>>>> But actually you already know my needs: I would like to make sysctls work in the
>>>>> way like sysfs does: i.e. content of files depends on mount maker -
>>>>> not viewer.
>>>>
>>>> What drives the desire to have sysctls depend on the mount maker?
>>>
>>> Because we can (will, actually) have nested fs root's for containers. IOW,
>>> container's root will be accessible from it's creator context. And I want to
>>> tune container's fs from creators context.
>>
>> Tuning the child context from the parent context is an entirely
>> reasonable thing to do. To affect a namespace that is not yours
>> the requirement is simply that we don't use current to lookup the
>> sysctl. So what I am proposing should work for your case.
>>
>
> Could you explain, what are you proposing?
> I still don't know any details about it.

I am proposing treating /proc/sys like /proc/net is currently treated.
See below.

>>>> Especially what drives that desire not to have it have a /proc/<pid>/sys
>>>> directory that reflects the sysctls for a given process.
>>>>
>>>
>>> This is not so important for me, where to access sysctl's. But I'm worrying
>>> about backward compatibility. IOW, I'm afraid of changing path
>>> "/proc/sys/sunprc/*" to "/proc/<pid>/sys/sunrpc". This would break a lot of
>>> user-space programs.
>>
>> The part that keeps it all working is by adding a symlink from /proc/sys
>> to /proc/self/sys. That technique has worked well for /proc/net, and I
>> don't expect there will be any problems with /proc/sys either. It is

>> possible but is very rare for the introduction of a symlink in a path
>> to cause problems.
>>
>
> Probably I don't understand you, but as I see it now, symlink to "/proc/self/"
> is unacceptable because of the following:
> 1) will be used current context (any) instead of desired one
(Using the current context is the desirable outcome for existing tools).
> 1) if CT has other pid namespace - then we just have broken link.

Assuming the process in question is not in the pid namespace available
to proc then yes you will indeed have a broken link. But a broken
link is only a problem for new applications that are doing something strange.

I am proposing treating /proc/sys like /proc/net has already been
treated. Aka move have the version of /proc/sys that relative to a
process be visible at: /proc/<pid>/sys, and with a compat symlink
from /proc/sys -> /proc/self/sys.

Just like has already been done with /proc/net.

Semantically this should be easy to understand, and about as backwards
compatible as it gets.

Eric

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by Stanislav Kinsbursky on Wed, 11 Jan 2012 18:02:16 GMT
View Forum Message <> Reply to Message

11.01.2012 21:21, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>>>>> Especially what drives that desire not to have it have a /proc/<pid>/sys
>>>>> directory that reflects the sysctls for a given process.
>>>>>
>>>>
>>>> This is not so important for me, where to access sysctl's. But I'm worrying
>>>> about backward compatibility. IOW, I'm afraid of changing path
>>>> "/proc/sys/sunprc/*" to "/proc/<pid>/sys/sunrpc". This would break a lot of
>>>> user-space programs.
>>>
>>> The part that keeps it all working is by adding a symlink from /proc/sys
>>> to /proc/self/sys. That technique has worked well for /proc/net, and I
>>> don't expect there will be any problems with /proc/sys either. It is
>>> possible but is very rare for the introduction of a symlink in a path
>>> to cause problems.
>>>
>>

>> Probably I don't understand you, but as I see it now, symlink to "/proc/self/"
>> is unacceptable because of the following:
>> 1) will be used current context (any) instead of desired one
> (Using the current context is the desirable outcome for existing tools).
>> 1) if CT has other pid namespace - then we just have broken link.
>
> Assuming the process in question is not in the pid namespace available
> to proc then yes you will indeed have a broken link. But a broken
> link is only a problem for new applications that are doing something strange.
>

I believe, that container is assuming to work in it's own network and pid
namespaces.
With your approach, if I'm not mistaken, container's /proc/net and /proc/sys
tunables will be unaccessible from parent environment. Or I'm wrong here?

> I am proposing treating /proc/sys like /proc/net has already been
> treated. Aka move have the version of /proc/sys that relative to a
> process be visible at: /proc/<pid>/sys, and with a compat symlink
> from /proc/sys -> /proc/self/sys.
>
> Just like has already been done with /proc/net.
>

1) On one hand it looks logical, that any nested dentries in /proc are tied to
pid namespace. But on the other hand we have a lot of tunables in /proc/net,
/proc/sys, etc. which have nothing with processes or whatever similar.
2) currently /proc processes directories (i.e. /proc/1/, etc) depends on mount
maker context. But /proc/sys and /proc/net doesn't. This looks weird and
despondently, from my pow. What do you think about it?

And what do you think about "conteinerization" of /proc contents in the way like
"sysfs" was done?
Implementing /proc "conteinerization" in this way can give us great flexibility.
For example, /proc/net (and /proc/sys/sunrpc) depends on mount owner net
namespace, /proc/sysvipc depends on mount owner ipc namespace, etc.
And this approach doesn't break backward compatibility as well.

> Semantically this should be easy to understand, and about as backwards
> compatible as it gets.
>
> Eric

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by ebiederm on Wed, 11 Jan 2012 19:34:38 GMT
View Forum Message <> Reply to Message

Stanislav Kinsbursky <skinsbursky@parallels.com> writes:

> 11.01.2012 21:21, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>>>>>> Especially what drives that desire not to have it have a /proc/<pid>/sys
>>>>>> directory that reflects the sysctls for a given process.
>>>>>>
>>>>>
>>>>> This is not so important for me, where to access sysctl's. But I'm worrying
>>>>> about backward compatibility. IOW, I'm afraid of changing path
>>>>> "/proc/sys/sunprc/*" to "/proc/<pid>/sys/sunrpc". This would break a lot of
>>>>> user-space programs.
>>>>
>>>> The part that keeps it all working is by adding a symlink from /proc/sys
>>>> to /proc/self/sys. That technique has worked well for /proc/net, and I
>>>> don't expect there will be any problems with /proc/sys either. It is
>>>> possible but is very rare for the introduction of a symlink in a path
>>>> to cause problems.
>>>>
>>>
>>> Probably I don't understand you, but as I see it now, symlink to "/proc/self/"
>>> is unacceptable because of the following:
>>> 1) will be used current context (any) instead of desired one
>> (Using the current context is the desirable outcome for existing tools).
>>> 1) if CT has other pid namespace - then we just have broken link.
>>
>> Assuming the process in question is not in the pid namespace available
>> to proc then yes you will indeed have a broken link. But a broken
>> link is only a problem for new applications that are doing something strange.
>>
>
> I believe, that container is assuming to work in it's own network and pid
> namespaces.
> With your approach, if I'm not mistaken, container's /proc/net and /proc/sys
> tunables will be unaccessible from parent environment. Or I'm wrong here?

Wrong.

>> I am proposing treating /proc/sys like /proc/net has already been
>> treated. Aka move have the version of /proc/sys that relative to a
>> process be visible at: /proc/<pid>/sys, and with a compat symlink
>> from /proc/sys -> /proc/self/sys.
>>
>> Just like has already been done with /proc/net.
>>
>

> 1) On one hand it looks logical, that any nested dentries in /proc are tied to
> pid namespace. But on the other hand we have a lot of tunables in /proc/net,
> /proc/sys, etc. which have nothing with processes or whatever similar.

Please stop and take a look at /proc/net. If your /proc/net is not a symlink please look at a modern kernel.

/proc/<pid>/net reflects the network namespace of the task in question.

> 2) currently /proc processes directories (i.e. /proc/1/, etc) depends on mount
> maker context. But /proc/sys and /proc/net doesn't. This looks weird and
> despondently, from my pow. What do you think about it?

Yep. Sysfs is weird. Ideally sysfs would display all devices all of the time but unfortunately that breaks backwards compatibility.

In proc we have the opportunity to display nearly everything all of the time and I think that opportunity is worth seizing.

Having to mount a filesystem simply because the designers of the filesystem were not creative enough to figure out how to display all of the information the filesystem is responsible for displaying without having namespace conflicts is unfortunate.

> And what do you think about "conteinerization" of /proc contents in the way like
> "sysfs" was done?

I think the way sysfs is done is a pain in the neck to use. Especially in the context of commands like "ip netns exec". With the sysfs model there is a lot of extra state to manage.

I totally agree that the way sysfs is done is much better than the way /proc/sys is done today. Looking at current can be limiting in the general case.

My current preference is the way /proc/net was done.

> Implementing /proc "conteinerization" in this way can give us great flexibility.
> For example, /proc/net (and /proc/sys/sunrpc) depends on mount owner net
> namespace, /proc/sysvipc depends on mount owner ipc namespace, etc.
> And this approach doesn't break backward compatibility as well.

The thing is /proc/net is already done.

All I see with making things like /proc/net depend on the context of the process that called mount is a need to call mount much more often.

Eric

Subject: Re: [PATCH 01/11] SYSCTL: export root and set handling routines
Posted by Stanislav Kinsbursky on Thu, 12 Jan 2012 09:17:16 GMT
View Forum Message <> Reply to Message

11.01.2012 23:36, Eric W. Biederman Ð¿Ð¸ÑˆÐµÑ‚:
>
> Please stop and take a look at /proc/net. If your /proc/net is not a
> symlink please look at a modern kernel.
>
> /proc/<pid>/net reflects the network namespace of the task in question.
>

Ok, I know that.
I know, that if some task with pid N is in other network namespace, then
/proc/<N>/net contents will differ to /proc/selt/net contents.

>> And what do you think about "conteinerization" of /proc contents in the way like
>> "sysfs" was done?
>
> I think the way sysfs is done is a pain in the neck to use. Especially
> in the context of commands like "ip netns exec". With the sysfs model
> there is a lot of extra state to manage.
>
> I totally agree that the way sysfs is done is much better than the way
> /proc/sys is done today. Looking at current can be limiting in the
> general case.
>
> My current preference is the way /proc/net was done.
>

Ok. But this approach still requires some additional data to manage in user
space. I.e. it's really easy to manage container's context using it's fs root,
because container's root is a part of initial configuration. But container's
processed pids numbers in parent context are unpredictable.

>> Implementing /proc "conteinerization" in this way can give us great flexibility.
>> For example, /proc/net (and /proc/sys/sunrpc) depends on mount owner net
>> namespace, /proc/sysvipc depends on mount owner ipc namespace, etc.
>> And this approach doesn't break backward compatibility as well.
>
> The thing is /proc/net is already done.
>
> All I see with making things like /proc/net depend on the context of the
> process that called mount is a need to call mount much more often.
>

/proc/net is a part or /proc. And /proc mount is called per container. So this
is just like it is.

I have some solution I mind, which looks quite simple to implement, doesn't require significant additional state to manage and suits my needs.
Please, consider this.
It's based on sysfs containerization approach, but simplified a lot.
Sysctl's (comparing to sysfs entries) entries are the same for all namespaces.
This actually means, that we don't need any additional infrastructure for managing dentries. All we need to know on read/write operations with sysctl's is the namespaces /proc was mounted from.

Thus if we:

1) replace /proc sb->s_fsdata content from pid_namespace to nsproxy and
2) add link to /proc sb to ctl_table and
3) add ns tag (pid, net, else or none) to ctl_table

then we will have all we need to manage sysctl's content in the way we want.
And looks like this approach doesn't break backward compatibility.

What do you think about it?

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH 00/11] SUNRPC: make sysctl per network namespcase context
Posted by Stanislav Kinsbursky on Tue, 07 Feb 2012 11:44:14 GMT
View Forum Message <> Reply to Message

Trond, what are your plans about this patch set?
Are you waiting for me and Eric to reach some decision, that suits both of us?

--
Best regards,
Stanislav Kinsbursky

Subject: Re: [PATCH 00/11] SUNRPC: make sysctl per network namespcase context
Posted by Myklebust, Trond on Tue, 07 Feb 2012 13:21:34 GMT
View Forum Message <> Reply to Message

On Tue, 2012-02-07 at 15:44 +0400, Stanislav Kinsbursky wrote:
> Trond, what are your plans about this patch set?
> Are you waiting for me and Eric to reach some decision, that suits both of us?

Yes. I'd like a consensus on the solution.

--
Trond Myklebust
Linux NFS client maintainer

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