Subject: [PATCH 0/12 net-2.6.26] icmp_socket namespacing
Posted by den on Fri, 29 Feb 2008 13:39:20 GMT
View Forum Message <> Reply to Message

This set of patches is targeted to create separate icmp_socket inside
each namespace. Both IPv4 and IPv6 codepaths are affected.

Though, in order to do this smoothly, a bit of optimisations are
performed. The kernel from now on will use sock rather than socket on
the ICMP send path.

Signed-off-by: Denis V. Lunev <den@openvz.org>

Subject: [PATCH 1/12 net-2.6.26] [INET]: Remove struct net_proto_family* from _init calls.
Posted by den on Fri, 29 Feb 2008 13:40:47 GMT
View Forum Message <> Reply to Message

struct net_proto_family* is not used in icmp[v6]_init, ndisc_init, igmp_init
and tcp_v4_init. Remove it.

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 include/linux/icmpv6.h | 2 +-
 include/net/icmp.h | 2 +-
 include/net/ndisc.h | 4 ++--
 include/net/tcp.h | 2 +-
 net/ipv4/af_inet.c | 4 ++--
 net/ipv4/icmp.c | 2 +-
 net/ipv4/tcp_ipv4.c | 2 +-
 net/ipv6/af_inet6.c | 6 +++---
 net/ipv6/icmp.c | 2 +-
 net/ipv6/mcast.c | 2 +-
 net/ipv6/ndisc.c | 2 +-
 11 files changed, 15 insertions(+), 15 deletions(-)

diff --git a/include/linux/icmpv6.h b/include/linux/icmpv6.h
index 7c5e981..8f86d6b 100644
--- a/include/linux/icmpv6.h
+++ b/include/linux/icmpv6.h
@@ -176,7 +176,7 @@ extern void   icmpv6_send(struct sk_buff *skb,

```
          __u32 info,
          struct net_device *dev);

-extern int    icmpv6_init(struct net_proto_family *ops);
+extern int    icmpv6_init(void);
 extern int    icmpv6_err_convert(int type, int code,
          int *err);
 extern void    icmpv6_cleanup(void);
diff --git a/include/net/icmp.h b/include/net/icmp.h
index 9f7ef3c..7bf714d 100644
--- a/include/net/icmp.h
+++ b/include/net/icmp.h
@@ -48,7 +48,7 @@ struct sk_buff;
 extern void icmp_send(struct sk_buff *skb_in,  int type, int code, __be32 info);
 extern int icmp_rcv(struct sk_buff *skb);
 extern int icmp_ioctl(struct sock *sk, int cmd, unsigned long arg);
-extern void icmp_init(struct net_proto_family *ops);
+extern void icmp_init(void);
 extern void icmp_out_count(unsigned char type);

 /* Move into dst.h ? */
diff --git a/include/net/ndisc.h b/include/net/ndisc.h
index 59b7062..5aedf32 100644
--- a/include/net/ndisc.h
+++ b/include/net/ndisc.h
@@ -77,7 +77,7 @@ struct nd_opt_hdr {
 } __attribute__((__packed__));


-extern int   ndisc_init(struct net_proto_family *ops);
+extern int   ndisc_init(void);

 extern void   ndisc_cleanup(void);

@@ -107,7 +107,7 @@ extern int   ndisc_mc_map(struct in6_addr *addr, char *buf, struct
net_device *d
 /*
  * IGMP
  */
-extern int   igmp6_init(struct net_proto_family *ops);
+extern int   igmp6_init(void);

 extern void   igmp6_cleanup(void);

diff --git a/include/net/tcp.h b/include/net/tcp.h
index 7de4ea3..ae9774b 100644
--- a/include/net/tcp.h
+++ b/include/net/tcp.h
```

```
@@ -1373,7 +1373,7 @@ struct tcp_request_sock_ops {
 #endif
 };

-extern void tcp_v4_init(struct net_proto_family *ops);
+extern void tcp_v4_init(void);
 extern void tcp_init(void);

 #endif /* _TCP_H */
diff --git a/net/ipv4/af_inet.c b/net/ipv4/af_inet.c
index c270080..a7a99ac 100644
--- a/net/ipv4/af_inet.c
+++ b/net/ipv4/af_inet.c
@@ -1415,7 +1415,7 @@ static int __init inet_init(void)

  ip_init();

- tcp_v4_init(&inet_family_ops);
+ tcp_v4_init();

  /* Setup TCP slab cache for open requests. */
  tcp_init();
@@ -1430,7 +1430,7 @@ static int __init inet_init(void)
   * Set the ICMP layer up
   */

- icmp_init(&inet_family_ops);
+ icmp_init();

  /*
   * Initialise the multicast router
diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index a13c074..98372db 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -1139,7 +1139,7 @@ static const struct icmp_control icmp_pointers[NR_ICMP_TYPES + 1]
= {
  },
 };

-void __init icmp_init(struct net_proto_family *ops)
+void __init icmp_init(void)
 {
  struct inet_sock *inet;
  int i;
diff --git a/net/ipv4/tcp_ipv4.c b/net/ipv4/tcp_ipv4.c
index 00156bf..256032a 100644
--- a/net/ipv4/tcp_ipv4.c
```

```
+++ b/net/ipv4/tcp_ipv4.c
@@ -2443,7 +2443,7 @@ struct proto tcp_prot = {
  REF_PROTO_INUSE(tcp)
 };

-void __init tcp_v4_init(struct net_proto_family *ops)
+void __init tcp_v4_init(void)
 {
  if (inet_csk_ctl_sock_create(&tcp_socket, PF_INET, SOCK_RAW,
         IPPROTO_TCP) < 0)
diff --git a/net/ipv6/af_inet6.c b/net/ipv6/af_inet6.c
index f0aa977..9869f87 100644
--- a/net/ipv6/af_inet6.c
+++ b/net/ipv6/af_inet6.c
@@ -808,13 +808,13 @@ static int __init inet6_init(void)
  if (err)
   goto sysctl_fail;
 #endif
- err = icmpv6_init(&inet6_family_ops);
+ err = icmpv6_init();
  if (err)
   goto icmp_fail;
- err = ndisc_init(&inet6_family_ops);
+ err = ndisc_init();
  if (err)
   goto ndisc_fail;
- err = igmp6_init(&inet6_family_ops);
+ err = igmp6_init();
  if (err)
   goto igmp_fail;
  err = ipv6_netfilter_init();
diff --git a/net/ipv6/icmp.c b/net/ipv6/icmp.c
index 121d517..b9b13a7 100644
--- a/net/ipv6/icmp.c
+++ b/net/ipv6/icmp.c
@@ -780,7 +780,7 @@ drop_no_count:
  */
 static struct lock_class_key icmpv6_socket_sk_dst_lock_key;

-int __init icmpv6_init(struct net_proto_family *ops)
+int __init icmpv6_init(void)
 {
  struct sock *sk;
  int err, i, j;
diff --git a/net/ipv6/mcast.c b/net/ipv6/mcast.c
index ab228d1..8ce894d 100644
--- a/net/ipv6/mcast.c
+++ b/net/ipv6/mcast.c
```

```
@@ -2597,7 +2597,7 @@ static const struct file_operations igmp6_mcf_seq_fops = {
 };
 #endif

-int __init igmp6_init(struct net_proto_family *ops)
+int __init igmp6_init(void)
 {
  struct ipv6_pinfo *np;
  struct sock *sk;
diff --git a/net/ipv6/ndisc.c b/net/ipv6/ndisc.c
index 0d33a7d..1fc33c8 100644
--- a/net/ipv6/ndisc.c
+++ b/net/ipv6/ndisc.c
@@ -1733,7 +1733,7 @@ static int ndisc_ifinfo_sysctl_strategy(ctl_table *ctl, int __user *name,

 #endif

-int __init ndisc_init(struct net_proto_family *ops)
+int __init ndisc_init(void)
 {
  struct ipv6_pinfo *np;
  struct sock *sk;
--
1.5.3.rc5
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 2/12 net-2.6.26] [ICMP]: Add return code to icmp_init.
Posted by den on Fri, 29 Feb 2008 13:40:48 GMT
View Forum Message <> Reply to Message

icmp_init could fail and this is normal for namespace other than initial.
So, the panic should be triggered only on init_net initialization path.

Additionally create rollback path for icmp_init as a separate function.
It will also be used later during namespace destruction.

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 include/net/icmp.h | 2 +-
 net/ipv4/af_inet.c | 3 ++-
 net/ipv4/icmp.c | 26 ++++++++++++++++++++++----
 3 files changed, 25 insertions(+), 6 deletions(-)

```
diff --git a/include/net/icmp.h b/include/net/icmp.h
index 7bf714d..faba64d 100644
--- a/include/net/icmp.h
+++ b/include/net/icmp.h
@@ -48,7 +48,7 @@ struct sk_buff;
 extern void icmp_send(struct sk_buff *skb_in,  int type, int code, __be32 info);
 extern int icmp_rcv(struct sk_buff *skb);
 extern int icmp_ioctl(struct sock *sk, int cmd, unsigned long arg);
-extern void icmp_init(void);
+extern int icmp_init(void);
 extern void icmp_out_count(unsigned char type);

 /* Move into dst.h ? */
diff --git a/net/ipv4/af_inet.c b/net/ipv4/af_inet.c
index a7a99ac..4f539bd 100644
--- a/net/ipv4/af_inet.c
+++ b/net/ipv4/af_inet.c
@@ -1430,7 +1430,8 @@ static int __init inet_init(void)
   * Set the ICMP layer up
   */

- icmp_init();
+ if (icmp_init() < 0)
+  panic("Failed to create the ICMP control socket.\n");

  /*
   * Initialise the multicast router
diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index 98372db..b345b3d 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -1139,19 +1139,32 @@ static const struct icmp_control icmp_pointers[NR_ICMP_TYPES +
1] = {
 };

-void __init icmp_init(void)
+static void __exit icmp_exit(void)
 {
- struct inet_sock *inet;
  int i;

  for_each_possible_cpu(i) {
-  int err;
+  struct socket *sock;
+
+  sock = per_cpu(__icmp_socket, i);
```

```
+  if (sock == NULL)
+   continue;
+  per_cpu(__icmp_socket, i) = NULL;
+  sock_release(sock);
+ }
+}

+int __init icmp_init(void)
+{
+ struct inet_sock *inet;
+ int i, err;
+
+ for_each_possible_cpu(i) {
   err = sock_create_kern(PF_INET, SOCK_RAW, IPPROTO_ICMP,
           &per_cpu(__icmp_socket, i));

   if (err < 0)
-   panic("Failed to create the ICMP control socket.\n");
+   goto fail;

   per_cpu(__icmp_socket, i)->sk->sk_allocation = GFP_ATOMIC;

@@ -1171,6 +1184,11 @@ void __init icmp_init(void)
    */
   per_cpu(__icmp_socket, i)->sk->sk_prot->unhash(per_cpu(__icmp_socket, i)->sk);
  }
+ return 0;
+
+fail:
+ icmp_exit();
+ return err;
 }

 EXPORT_SYMBOL(icmp_err_convert);
--
1.5.3.rc5
```


Subject: [PATCH 3/12 net-2.6.26] [ICMP]: Optimize icmp_socket usage.
Posted by den on Fri, 29 Feb 2008 13:40:49 GMT
View Forum Message <> Reply to Message

Use this macro only once in a function to save a bit of space.

```
add/remove: 0/0 grow/shrink: 0/3 up/down: 0/-98 (-98)
function                                     old     new   delta
icmp_reply                                   562     561      -1
icmp_push_reply                              305     258     -47
icmp_init                                    273     223     -50

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 net/ipv4/icmp.c |   29 ++++++++++++++++++++------------
 1 files changed, 17 insertions(+), 12 deletions(-)

diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index b345b3d..831b6ad 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -346,19 +346,21 @@ static int icmp_glue_bits(void *from, char *to, int offset, int len, int odd,
 static void icmp_push_reply(struct icmp_bxm *icmp_param,
        struct ipcm_cookie *ipc, struct rtable *rt)
 {
+ struct sock *sk;
  struct sk_buff *skb;

- if (ip_append_data(icmp_socket->sk, icmp_glue_bits, icmp_param,
+ sk = icmp_socket->sk;
+ if (ip_append_data(sk, icmp_glue_bits, icmp_param,
        icmp_param->data_len+icmp_param->head_len,
        icmp_param->head_len,
        ipc, rt, MSG_DONTWAIT) < 0)
-  ip_flush_pending_frames(icmp_socket->sk);
- else if ((skb = skb_peek(&icmp_socket->sk->sk_write_queue)) != NULL) {
+  ip_flush_pending_frames(sk);
+ else if ((skb = skb_peek(&sk->sk_write_queue)) != NULL) {
   struct icmphdr *icmph = icmp_hdr(skb);
   __wsum csum = 0;
   struct sk_buff *skb1;

-  skb_queue_walk(&icmp_socket->sk->sk_write_queue, skb1) {
+  skb_queue_walk(&sk->sk_write_queue, skb1) {
    csum = csum_add(csum, skb1->csum);
   }
   csum = csum_partial_copy_nocheck((void *)&icmp_param->data,
@@ -366,7 +368,7 @@ static void icmp_push_reply(struct icmp_bxm *icmp_param,
        icmp_param->head_len, csum);
   icmph->checksum = csum_fold(csum);
   skb->ip_summed = CHECKSUM_NONE;
-  ip_push_pending_frames(icmp_socket->sk);
```

```
+  ip_push_pending_frames(sk);
  }
 }

@@ -1156,25 +1158,28 @@ static void __exit icmp_exit(void)

 int __init icmp_init(void)
 {
- struct inet_sock *inet;
  int i, err;

  for_each_possible_cpu(i) {
-  err = sock_create_kern(PF_INET, SOCK_RAW, IPPROTO_ICMP,
-         &per_cpu(__icmp_socket, i));
+  struct sock *sk;
+  struct socket *sock;
+  struct inet_sock *inet;

+  err = sock_create_kern(PF_INET, SOCK_RAW, IPPROTO_ICMP, &sock);
   if (err < 0)
    goto fail;

-  per_cpu(__icmp_socket, i)->sk->sk_allocation = GFP_ATOMIC;
+  per_cpu(__icmp_socket, i) = sock;
+  sk = sock->sk;
+  sk->sk_allocation = GFP_ATOMIC;

   /* Enough space for 2 64K ICMP packets, including
    * sk_buff struct overhead.
    */
-  per_cpu(__icmp_socket, i)->sk->sk_sndbuf =
+  sk->sk_sndbuf =
    (2 * ((64 * 1024) + sizeof(struct sk_buff)));

-  inet = inet_sk(per_cpu(__icmp_socket, i)->sk);
+  inet = inet_sk(sk);
   inet->uc_ttl = -1;
   inet->pmtudisc = IP_PMTUDISC_DONT;

@@ -1182,7 +1187,7 @@ int __init icmp_init(void)
    * see it, we do not wish this socket to see incoming
    * packets.
    */
-  per_cpu(__icmp_socket, i)->sk->sk_prot->unhash(per_cpu(__icmp_socket, i)->sk);
+  sk->sk_prot->unhash(sk);
  }
  return 0;
```

--
1.5.3.rc5

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 4/12 net-2.6.26] [ICMP]: Store sock rather than socket for ICMP flow control.
Posted by den on Fri, 29 Feb 2008 13:40:50 GMT
View Forum Message <> Reply to Message

Basically, there is no difference, what to store: socket or sock. Though,
sock looks better as there will be 1 less dereferrence on the fast path.

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 net/ipv4/icmp.c | 27 ++++++++++++++----------
 net/ipv6/icmp.c | 25 ++++++++++++++----------
 2 files changed, 26 insertions(+), 26 deletions(-)

diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index 831b6ad..3a4da43 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -229,14 +229,14 @@ static const struct icmp_control icmp_pointers[NR_ICMP_TYPES+1];
  *
  * On SMP we have one ICMP socket per-cpu.
  */
-static DEFINE_PER_CPU(struct socket *, __icmp_socket) = NULL;
-#define icmp_socket __get_cpu_var(__icmp_socket)
+static DEFINE_PER_CPU(struct sock *, __icmp_sk) = NULL;
+#define icmp_sk __get_cpu_var(__icmp_sk)

 static inline int icmp_xmit_lock(void)
 {
  local_bh_disable();

- if (unlikely(!spin_trylock(&icmp_socket->sk->sk_lock.slock))) {
+ if (unlikely(!spin_trylock(&icmp_sk->sk_lock.slock))) {
    /* This can happen if the output path signals a
     * dst_link_failure() for an outgoing ICMP packet.
     */
@@ -248,7 +248,7 @@ static inline int icmp_xmit_lock(void)

```
 static inline void icmp_xmit_unlock(void)
 {
- spin_unlock_bh(&icmp_socket->sk->sk_lock.slock);
+ spin_unlock_bh(&icmp_sk->sk_lock.slock);
 }

 /*
@@ -349,7 +349,7 @@ static void icmp_push_reply(struct icmp_bxm *icmp_param,
  struct sock *sk;
  struct sk_buff *skb;

- sk = icmp_socket->sk;
+ sk = icmp_sk;
  if (ip_append_data(sk, icmp_glue_bits, icmp_param,
       icmp_param->data_len+icmp_param->head_len,
       icmp_param->head_len,
@@ -378,7 +378,7 @@ static void icmp_push_reply(struct icmp_bxm *icmp_param,

 static void icmp_reply(struct icmp_bxm *icmp_param, struct sk_buff *skb)
 {
- struct sock *sk = icmp_socket->sk;
+ struct sock *sk = icmp_sk;
  struct inet_sock *inet = inet_sk(sk);
  struct ipcm_cookie ipc;
  struct rtable *rt = (struct rtable *)skb->dst;
@@ -546,7 +546,7 @@ void icmp_send(struct sk_buff *skb_in, int type, int code, __be32 info)
  icmp_param.data.icmph.checksum  = 0;
  icmp_param.skb   = skb_in;
  icmp_param.offset = skb_network_offset(skb_in);
- inet_sk(icmp_socket->sk)->tos = tos;
+ inet_sk(icmp_sk)->tos = tos;
  ipc.addr = iph->saddr;
  ipc.opt = &icmp_param.replyopts;

@@ -1146,13 +1146,13 @@ static void __exit icmp_exit(void)
  int i;

  for_each_possible_cpu(i) {
-  struct socket *sock;
+  struct sock *sk;

-  sock = per_cpu(__icmp_socket, i);
-  if (sock == NULL)
+  sk = per_cpu(__icmp_sk, i);
+  if (sk == NULL)
     continue;
-  per_cpu(__icmp_socket, i) = NULL;
-  sock_release(sock);
```

```
+  per_cpu(__icmp_sk, i) = NULL;
+  sock_release(sk->sk_socket);
  }
 }

@@ -1169,8 +1169,7 @@ int __init icmp_init(void)
   if (err < 0)
    goto fail;

-  per_cpu(__icmp_socket, i) = sock;
-  sk = sock->sk;
+  per_cpu(__icmp_sk, i) = sk = sock->sk;
   sk->sk_allocation = GFP_ATOMIC;

   /* Enough space for 2 64K ICMP packets, including
diff --git a/net/ipv6/icmp.c b/net/ipv6/icmp.c
index b9b13a7..875bdc7 100644
--- a/net/ipv6/icmp.c
+++ b/net/ipv6/icmp.c
@@ -80,8 +80,8 @@ EXPORT_SYMBOL(icmpv6msg_statistics);
  *
  * On SMP we have one ICMP socket per-cpu.
  */
-static DEFINE_PER_CPU(struct socket *, __icmpv6_socket) = NULL;
-#define icmpv6_socket __get_cpu_var(__icmpv6_socket)
+static DEFINE_PER_CPU(struct sock *, __icmpv6_sk) = NULL;
+#define icmpv6_sk __get_cpu_var(__icmpv6_sk)

 static int icmpv6_rcv(struct sk_buff *skb);

@@ -94,7 +94,7 @@ static __inline__ int icmpv6_xmit_lock(void)
 {
  local_bh_disable();

- if (unlikely(!spin_trylock(&icmpv6_socket->sk->sk_lock.slock))) {
+ if (unlikely(!spin_trylock(&icmpv6_sk->sk_lock.slock))) {
   /* This can happen if the output path (f.e. SIT or
    * ip6ip6 tunnel) signals dst_link_failure() for an
    * outgoing ICMP6 packet.
@@ -107,7 +107,7 @@ static __inline__ int icmpv6_xmit_lock(void)

 static __inline__ void icmpv6_xmit_unlock(void)
 {
- spin_unlock_bh(&icmpv6_socket->sk->sk_lock.slock);
+ spin_unlock_bh(&icmpv6_sk->sk_lock.slock);
 }

 /*
```

```
@@ -392,7 +392,7 @@ void icmpv6_send(struct sk_buff *skb, int type, int code, __u32 info,
  if (icmpv6_xmit_lock())
   return;

- sk = icmpv6_socket->sk;
+ sk = icmpv6_sk;
  np = inet6_sk(sk);

  if (!icmpv6_xrlim_allow(sk, type, &fl))
@@ -538,7 +538,7 @@ static void icmpv6_echo_reply(struct sk_buff *skb)
  if (icmpv6_xmit_lock())
   return;

- sk = icmpv6_socket->sk;
+ sk = icmpv6_sk;
  np = inet6_sk(sk);

  if (!fl.oif && ipv6_addr_is_multicast(&fl.fl6_dst))
@@ -776,7 +776,7 @@ drop_no_count:
 }

 /*
- * Special lock-class for __icmpv6_socket:
+ * Special lock-class for __icmpv6_sk:
  */
 static struct lock_class_key icmpv6_socket_sk_dst_lock_key;

@@ -786,8 +786,9 @@ int __init icmpv6_init(void)
  int err, i, j;

  for_each_possible_cpu(i) {
+  struct socket *sock;
   err = sock_create_kern(PF_INET6, SOCK_RAW, IPPROTO_ICMPV6,
-          &per_cpu(__icmpv6_socket, i));
+          &sock);
   if (err < 0) {
    printk(KERN_ERR
           "Failed to initialize the ICMP6 control socket "
@@ -796,12 +797,12 @@ int __init icmpv6_init(void)
    goto fail;
   }

-  sk = per_cpu(__icmpv6_socket, i)->sk;
+  per_cpu(__icmpv6_sk, i) = sk = sock->sk;
   sk->sk_allocation = GFP_ATOMIC;
   /*
    * Split off their lock-class, because sk->sk_dst_lock
    * gets used from softirqs, which is safe for
```

```
-   * __icmpv6_socket (because those never get directly used
+   * __icmpv6_sk (because those never get directly used
     * via userspace syscalls), but unsafe for normal sockets.
     */
   lockdep_set_class(&sk->sk_dst_lock,
@@ -829,7 +830,7 @@ int __init icmpv6_init(void)
  for (j = 0; j < i; j++) {
   if (!cpu_possible(j))
    continue;
-  sock_release(per_cpu(__icmpv6_socket, j));
+  sock_release(per_cpu(__icmpv6_sk, j)->sk_socket);
  }

  return err;
@@ -840,7 +841,7 @@ void icmpv6_cleanup(void)
  int i;

  for_each_possible_cpu(i) {
-  sock_release(per_cpu(__icmpv6_socket, i));
+  sock_release(per_cpu(__icmpv6_sk, i)->sk_socket);
  }
  inet6_del_protocol(&icmpv6_protocol, IPPROTO_ICMPV6);
 }
--
1.5.3.rc5
```

Subject: [PATCH 5/12 net-2.6.26] [ICMP]: Pass proper ICMP socket into icmp(v6)_xmit_(un)lock.

Posted by den on Fri, 29 Feb 2008 13:40:51 GMT

View Forum Message <> Reply to Message

We have to get socket lock inside icmp(v6)_xmit_lock/unlock. The socket is get from global variable now. When this code became namespaces, one should pass a namespace and get socket from it.

Though, above is useless. Socket is available in the caller, just pass it inside. This saves a bit of code now and saves more later.

```
add/remove: 0/0 grow/shrink: 1/3 up/down: 1/-169 (-168)
function                                     old     new   delta
icmp_rcv                                     718     719      +1
icmpv6_rcv                                  2343    2303     -40
```

```
icmp_send                       1566    1518     -48
icmp_reply                      549     468     -81

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 net/ipv4/icmp.c |   19 ++++++++++---------
 net/ipv6/icmp.c |   24 ++++++++++++++----------
 2 files changed, 22 insertions(+), 21 deletions(-)

diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index 3a4da43..9bcf263 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -232,11 +232,11 @@ static const struct icmp_control icmp_pointers[NR_ICMP_TYPES+1];
 static DEFINE_PER_CPU(struct sock *, __icmp_sk) = NULL;
 #define icmp_sk  __get_cpu_var(__icmp_sk)

-static inline int icmp_xmit_lock(void)
+static inline int icmp_xmit_lock(struct sock *sk)
 {
  local_bh_disable();

- if (unlikely(!spin_trylock(&icmp_sk->sk_lock.slock))) {
+ if (unlikely(!spin_trylock(&sk->sk_lock.slock))) {
   /* This can happen if the output path signals a
    * dst_link_failure() for an outgoing ICMP packet.
    */
@@ -246,9 +246,9 @@ static inline int icmp_xmit_lock(void)
  return 0;
 }

-static inline void icmp_xmit_unlock(void)
+static inline void icmp_xmit_unlock(struct sock *sk)
 {
- spin_unlock_bh(&icmp_sk->sk_lock.slock);
+ spin_unlock_bh(&sk->sk_lock.slock);
 }

 /*
@@ -387,7 +387,7 @@ static void icmp_reply(struct icmp_bxm *icmp_param, struct sk_buff
*skb)
  if (ip_options_echo(&icmp_param->replyopts, skb))
   return;

- if (icmp_xmit_lock())
+ if (icmp_xmit_lock(sk))
   return;
```

```
  icmp_param->data.icmph.checksum = 0;
@@ -415,7 +415,7 @@ static void icmp_reply(struct icmp_bxm *icmp_param, struct sk_buff
*skb)
   icmp_push_reply(icmp_param, &ipc, rt);
  ip_rt_put(rt);
 out_unlock:
- icmp_xmit_unlock();
+ icmp_xmit_unlock(sk);
 }


@@ -440,6 +440,7 @@ void icmp_send(struct sk_buff *skb_in, int type, int code, __be32 info)
  __be32 saddr;
  u8  tos;
  struct net *net;
+ struct sock *sk = icmp_sk;

  if (!rt)
   goto out;
@@ -507,7 +508,7 @@ void icmp_send(struct sk_buff *skb_in, int type, int code, __be32 info)
   }
  }

- if (icmp_xmit_lock())
+ if (icmp_xmit_lock(sk))
   return;

  /*
@@ -546,7 +547,7 @@ void icmp_send(struct sk_buff *skb_in, int type, int code, __be32 info)
  icmp_param.data.icmph.checksum  = 0;
  icmp_param.skb   = skb_in;
  icmp_param.offset = skb_network_offset(skb_in);
- inet_sk(icmp_sk)->tos = tos;
+ inet_sk(sk)->tos = tos;
  ipc.addr = iph->saddr;
  ipc.opt = &icmp_param.replyopts;

@@ -654,7 +655,7 @@ route_done:
 ende:
  ip_rt_put(rt);
 out_unlock:
- icmp_xmit_unlock();
+ icmp_xmit_unlock(sk);
 out:;
 }

diff --git a/net/ipv6/icmp.c b/net/ipv6/icmp.c
```

```
index 875bdc7..18f220a 100644
--- a/net/ipv6/icmp.c
+++ b/net/ipv6/icmp.c
@@ -90,11 +90,11 @@ static struct inet6_protocol icmpv6_protocol = {
  .flags  = INET6_PROTO_NOPOLICY|INET6_PROTO_FINAL,
 };

-static __inline__ int icmpv6_xmit_lock(void)
+static __inline__ int icmpv6_xmit_lock(struct sock *sk)
 {
  local_bh_disable();

- if (unlikely(!spin_trylock(&icmpv6_sk->sk_lock.slock))) {
+ if (unlikely(!spin_trylock(&sk->sk_lock.slock))) {
    /* This can happen if the output path (f.e. SIT or
     * ip6ip6 tunnel) signals dst_link_failure() for an
     * outgoing ICMP6 packet.
@@ -105,9 +105,9 @@ static __inline__ int icmpv6_xmit_lock(void)
  return 0;
 }

-static __inline__ void icmpv6_xmit_unlock(void)
+static __inline__ void icmpv6_xmit_unlock(struct sock *sk)
 {
- spin_unlock_bh(&icmpv6_sk->sk_lock.slock);
+ spin_unlock_bh(&sk->sk_lock.slock);
 }

 /*
@@ -389,12 +389,12 @@ void icmpv6_send(struct sk_buff *skb, int type, int code, __u32 info,
  fl.fl_icmp_code = code;
  security_skb_classify_flow(skb, &fl);

- if (icmpv6_xmit_lock())
-  return;
-
  sk = icmpv6_sk;
  np = inet6_sk(sk);

+ if (icmpv6_xmit_lock(sk))
+  return;
+
  if (!icmpv6_xrlim_allow(sk, type, &fl))
   goto out;

@@ -498,7 +498,7 @@ out_put:
 out_dst_release:
  dst_release(dst);
```

```
 out:
- icmpv6_xmit_unlock();
+ icmpv6_xmit_unlock(sk);
 }

 EXPORT_SYMBOL(icmpv6_send);
@@ -535,12 +535,12 @@ static void icmpv6_echo_reply(struct sk_buff *skb)
  fl.fl_icmp_type = ICMPV6_ECHO_REPLY;
  security_skb_classify_flow(skb, &fl);

- if (icmpv6_xmit_lock())
-  return;
-
  sk = icmpv6_sk;
  np = inet6_sk(sk);

+ if (icmpv6_xmit_lock(sk))
+  return;
+
  if (!fl.oif && ipv6_addr_is_multicast(&fl.fl6_dst))
   fl.oif = np->mcast_oif;

@@ -584,7 +584,7 @@ out_put:
   in6_dev_put(idev);
  dst_release(dst);
 out:
- icmpv6_xmit_unlock();
+ icmpv6_xmit_unlock(sk);
 }

 static void icmpv6_notify(struct sk_buff *skb, int type, int code, __be32 info)
--
1.5.3.rc5
```

Subject: [PATCH 6/12 net-2.6.26] [ICMP]: Allocate data for __icmp(v6)_sk dynamically.

Posted by den on Fri, 29 Feb 2008 13:40:52 GMT

View Forum Message <> Reply to Message

Own __icmp(v6)_sk should be present in each namespace. So, it should be allocated dynamically. Though, alloc_percpu does not fit the case as it implies additional dereferrence for no bonus.

Allocate data for pointers just like __percpu_alloc_mask does and place
pointers to struct sock into this array.

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 net/ipv4/icmp.c | 15 ++++++++++-----
 net/ipv6/icmp.c | 14 +++++++++-----
 2 files changed, 19 insertions(+), 10 deletions(-)

```
diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index 9bcf263..7c62a0d 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -229,8 +229,8 @@ static const struct icmp_control icmp_pointers[NR_ICMP_TYPES+1];
  *
  * On SMP we have one ICMP socket per-cpu.
  */
-static DEFINE_PER_CPU(struct sock *, __icmp_sk) = NULL;
-#define icmp_sk  __get_cpu_var(__icmp_sk)
+static struct sock **__icmp_sk = NULL;
+#define icmp_sk  (__icmp_sk[smp_processor_id()])

 static inline int icmp_xmit_lock(struct sock *sk)
 {
@@ -1149,18 +1149,23 @@ static void __exit icmp_exit(void)
  for_each_possible_cpu(i) {
   struct sock *sk;

-  sk = per_cpu(__icmp_sk, i);
+  sk = __icmp_sk[i];
   if (sk == NULL)
    continue;
-  per_cpu(__icmp_sk, i) = NULL;
   sock_release(sk->sk_socket);
  }
+ kfree(__icmp_sk);
+ __icmp_sk = NULL;
 }

 int __init icmp_init(void)
 {
  int i, err;

+ __icmp_sk = kzalloc(nr_cpu_ids * sizeof(struct sock *), GFP_KERNEL);
+ if (__icmp_sk == NULL)
+  return -ENOMEM;
```

```
+
  for_each_possible_cpu(i) {
   struct sock *sk;
   struct socket *sock;
@@ -1170,7 +1175,7 @@ int __init icmp_init(void)
   if (err < 0)
    goto fail;

-  per_cpu(__icmp_sk, i) = sk = sock->sk;
+  __icmp_sk[i] = sk = sock->sk;
   sk->sk_allocation = GFP_ATOMIC;

   /* Enough space for 2 64K ICMP packets, including
diff --git a/net/ipv6/icmp.c b/net/ipv6/icmp.c
index 18f220a..3368f32 100644
--- a/net/ipv6/icmp.c
+++ b/net/ipv6/icmp.c
@@ -80,8 +80,8 @@ EXPORT_SYMBOL(icmpv6msg_statistics);
  *
  * On SMP we have one ICMP socket per-cpu.
  */
-static DEFINE_PER_CPU(struct sock *, __icmpv6_sk) = NULL;
-#define icmpv6_sk __get_cpu_var(__icmpv6_sk)
+static struct sock **__icmpv6_sk = NULL;
+#define icmpv6_sk (__icmpv6_sk[smp_processor_id()])

 static int icmpv6_rcv(struct sk_buff *skb);

@@ -785,6 +785,10 @@ int __init icmpv6_init(void)
  struct sock *sk;
  int err, i, j;

+ __icmpv6_sk = kzalloc(nr_cpu_ids * sizeof(struct sock *), GFP_KERNEL);
+ if (__icmpv6_sk == NULL)
+  return -ENOMEM;
+
  for_each_possible_cpu(i) {
   struct socket *sock;
   err = sock_create_kern(PF_INET6, SOCK_RAW, IPPROTO_ICMPV6,
@@ -797,7 +801,7 @@ int __init icmpv6_init(void)
    goto fail;
   }

-  per_cpu(__icmpv6_sk, i) = sk = sock->sk;
+  __icmpv6_sk[i] = sk = sock->sk;
   sk->sk_allocation = GFP_ATOMIC;
   /*
    * Split off their lock-class, because sk->sk_dst_lock
```

```
@@ -830,7 +834,7 @@ int __init icmpv6_init(void)
  for (j = 0; j < i; j++) {
   if (!cpu_possible(j))
    continue;
-  sock_release(per_cpu(__icmpv6_sk, j)->sk_socket);
+  sock_release(__icmpv6_sk[j]->sk_socket);
  }

  return err;
@@ -841,7 +845,7 @@ void icmpv6_cleanup(void)
  int i;

  for_each_possible_cpu(i) {
-  sock_release(per_cpu(__icmpv6_sk, i)->sk_socket);
+  sock_release(__icmpv6_sk[i]->sk_socket);
  }
  inet6_del_protocol(&icmpv6_protocol, IPPROTO_ICMPV6);
 }
--
1.5.3.rc5
```

Subject: [PATCH 7/12 net-2.6.26] No need for a separate __netlink_release call.
Posted by den on Fri, 29 Feb 2008 13:40:53 GMT
View Forum Message <> Reply to Message

Merge it to netlink_kernel_release.

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>

```
---
 net/netlink/af_netlink.c |   30 ++++++++++++++----------------
 1 files changed, 12 insertions(+), 18 deletions(-)

diff --git a/net/netlink/af_netlink.c b/net/netlink/af_netlink.c
index 1ab0da2..e6b636d 100644
--- a/net/netlink/af_netlink.c
+++ b/net/netlink/af_netlink.c
@@ -1344,22 +1344,6 @@ static void netlink_data_ready(struct sock *sk, int len)
  * queueing.
  */

-static void __netlink_release(struct sock *sk)
```

```
-{
- /*
-  * Last sock_put should drop referrence to sk->sk_net. It has already
-  * been dropped in netlink_kernel_create. Taking referrence to stopping
-  * namespace is not an option.
-  * Take referrence to a socket to remove it from netlink lookup table
-  * _alive_ and after that destroy it in the context of init_net.
-  */
-
- sock_hold(sk);
- sock_release(sk->sk_socket);
- sk->sk_net = get_net(&init_net);
- sock_put(sk);
-}
-
 struct sock *
 netlink_kernel_create(struct net *net, int unit, unsigned int groups,
         void (*input)(struct sk_buff *skb),
@@ -1424,7 +1408,7 @@ netlink_kernel_create(struct net *net, int unit, unsigned int groups,

 out_sock_release:
   kfree(listeners);
- __netlink_release(sk);
+ netlink_kernel_release(sk);
   return NULL;

 out_sock_release_nosk:
@@ -1437,10 +1421,20 @@ EXPORT_SYMBOL(netlink_kernel_create);
 void
 netlink_kernel_release(struct sock *sk)
 {
+ /*
+  * Last sock_put should drop referrence to sk->sk_net. It has already
+  * been dropped in netlink_kernel_create. Taking referrence to stopping
+  * namespace is not an option.
+  * Take referrence to a socket to remove it from netlink lookup table
+  * _alive_ and after that destroy it in the context of init_net.
+  */
   if (sk == NULL || sk->sk_socket == NULL)
     return;

- __netlink_release(sk);
+ sock_hold(sk);
+ sock_release(sk->sk_socket);
+ sk->sk_net = get_net(&init_net);
+ sock_put(sk);
 }
 EXPORT_SYMBOL(netlink_kernel_release);
```

--
1.5.3.rc5

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 8/12 net-2.6.26] Make netlink_kernel_release publically available as sk_release_kernel.
Posted by den on Fri, 29 Feb 2008 13:40:54 GMT
View Forum Message <> Reply to Message

This staff will be needed for non-netlink kernel sockets, which should also
not pin a namespace like tcp_socket and icmp_socket.

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 include/net/sock.h | 13 ++++++++++++
 net/core/sock.c | 18 ++++++++++++++++++
 net/netlink/af_netlink.c | 18 ++++++----------------
 3 files changed, 33 insertions(+), 16 deletions(-)

diff --git a/include/net/sock.h b/include/net/sock.h
index fd98760..39112e7 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -850,6 +850,7 @@ extern struct sock *sk_alloc(struct net *net, int family,
        gfp_t priority,
        struct proto *prot);
 extern void   sk_free(struct sock *sk);
+extern void   sk_release_kernel(struct sock *sk);
 extern struct sock *sk_clone(const struct sock *sk,
        const gfp_t priority);

@@ -1333,6 +1334,18 @@ static inline void sk_eat_skb(struct sock *sk, struct sk_buff *skb, int copied_e
 }
 #endif

+/*
+ * Kernel sockets, f.e. rtnl or icmp_socket, are a part of a namespace.
+ * They should not hold a referrence to a namespace in order to allow
+ * to stop it.
+ * Sockets after sk_change_net should be released using sk_release_kernel

```
+ */
+static inline void sk_change_net(struct sock *sk, struct net *net)
+{
+ put_net(sk->sk_net);
+ sk->sk_net = net;
+}
+
 extern void sock_enable_timestamp(struct sock *sk);
 extern int sock_get_timestamp(struct sock *, struct timeval __user *);
 extern int sock_get_timestampns(struct sock *, struct timespec __user *);
diff --git a/net/core/sock.c b/net/core/sock.c
index 09cb3a7..c71b645 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -987,6 +987,24 @@ void sk_free(struct sock *sk)
  sk_prot_free(sk->sk_prot_creator, sk);
 }

+/*
+ * Last sock_put should drop referrence to sk->sk_net. It has already
+ * been dropped in sk_change_net. Taking referrence to stopping namespace
+ * is not an option.
+ * Take referrence to a socket to remove it from hash _alive_ and after that
+ * destroy it in the context of init_net.
+ */
+void sk_release_kernel(struct sock *sk)
+{
+ if (sk == NULL || sk->sk_socket == NULL)
+  return;
+
+ sock_hold(sk);
+ sock_release(sk->sk_socket);
+ sk->sk_net = get_net(&init_net);
+ sock_put(sk);
+}
+
 struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
 {
  struct sock *newsk;
diff --git a/net/netlink/af_netlink.c b/net/netlink/af_netlink.c
index e6b636d..524e826 100644
--- a/net/netlink/af_netlink.c
+++ b/net/netlink/af_netlink.c
@@ -1372,8 +1372,7 @@ netlink_kernel_create(struct net *net, int unit, unsigned int groups,
   goto out_sock_release_nosk;

  sk = sock->sk;
- put_net(sk->sk_net);
```

```
- sk->sk_net = net;
+ sk_change_net(sk, net);

  if (groups < 32)
   groups = 32;
@@ -1421,20 +1420,7 @@ EXPORT_SYMBOL(netlink_kernel_create);
 void
 netlink_kernel_release(struct sock *sk)
 {
- /*
-  * Last sock_put should drop referrence to sk->sk_net. It has already
-  * been dropped in netlink_kernel_create. Taking referrence to stopping
-  * namespace is not an option.
-  * Take referrence to a socket to remove it from netlink lookup table
-  * _alive_ and after that destroy it in the context of init_net.
-  */
- if (sk == NULL || sk->sk_socket == NULL)
-  return;
-
- sock_hold(sk);
- sock_release(sk->sk_socket);
- sk->sk_net = get_net(&init_net);
- sock_put(sk);
+ sk_release_kernel(sk);
 }
 EXPORT_SYMBOL(netlink_kernel_release);

--
1.5.3.rc5
```

Subject: [PATCH 9/12 net-2.6.26] [NETNS]: icmp(v6)_sk should not pin a namespace.

Posted by den on Fri, 29 Feb 2008 13:40:55 GMT

View Forum Message <> Reply to Message

So, change icmp(v6)_sk creation/disposal to the scheme used in the netlink
for rtnl, i.e. create a socket in the context of the init_net and
assign the namespace without getting a referrence later.

Also use sk_release_kernel instead of sock_release to properly destroy
such sockets.

```
Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 net/ipv4/icmp.c |   12 ++++++--------
 net/ipv6/icmp.c |   11 +++++++------
 2 files changed, 9 insertions(+), 14 deletions(-)

diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index 7c62a0d..97d97ad 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -1146,14 +1146,8 @@ static void __exit icmp_exit(void)
 {
  int i;

- for_each_possible_cpu(i) {
-  struct sock *sk;
-
-  sk = __icmp_sk[i];
-  if (sk == NULL)
-   continue;
-  sock_release(sk->sk_socket);
- }
+ for_each_possible_cpu(i)
+  sk_release_kernel(__icmp_sk[i]);
  kfree(__icmp_sk);
  __icmp_sk = NULL;
 }
@@ -1176,6 +1170,8 @@ int __init icmp_init(void)
    goto fail;

   __icmp_sk[i] = sk = sock->sk;
+  sk_change_net(sk, &init_net);
+
   sk->sk_allocation = GFP_ATOMIC;

   /* Enough space for 2 64K ICMP packets, including
diff --git a/net/ipv6/icmp.c b/net/ipv6/icmp.c
index 3368f32..7341d79 100644
--- a/net/ipv6/icmp.c
+++ b/net/ipv6/icmp.c
@@ -802,6 +802,8 @@ int __init icmpv6_init(void)
   }

   __icmpv6_sk[i] = sk = sock->sk;
+   sk_change_net(sk, &init_net);
+
   sk->sk_allocation = GFP_ATOMIC;
```

```
   /*
    * Split off their lock-class, because sk->sk_dst_lock
@@ -831,11 +833,8 @@ int __init icmpv6_init(void)
  return 0;

  fail:
- for (j = 0; j < i; j++) {
-  if (!cpu_possible(j))
-   continue;
-  sock_release(__icmpv6_sk[j]->sk_socket);
- }
+ for (j = 0; j < i; j++)
+  sk_release_kernel(__icmpv6_sk[j]);

  return err;
 }
@@ -845,7 +844,7 @@ void icmpv6_cleanup(void)
  int i;

  for_each_possible_cpu(i) {
-  sock_release(__icmpv6_sk[i]->sk_socket);
+  sk_release_kernel(__icmpv6_sk[i]);
  }
  inet6_del_protocol(&icmpv6_protocol, IPPROTO_ICMPV6);
 }
--
1.5.3.rc5
```

____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 10/12] [NETNS]: Make icmp_sk per namespace.
Posted by den on Fri, 29 Feb 2008 13:40:56 GMT
View Forum Message <> Reply to Message

All preparations are done. Now just add a hook to perform an initialization
on namespace startup and replace icmp_sk macro with proper inline call.

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>

```
---
 include/net/netns/ipv4.h |    2 +
 net/ipv4/icmp.c          |   49 +++++++++++++++++++++++++++++++++----------------
 2 files changed, 34 insertions(+), 17 deletions(-)
```

```
diff --git a/include/net/netns/ipv4.h b/include/net/netns/ipv4.h
index a9b4f60..504fde1 100644
--- a/include/net/netns/ipv4.h
+++ b/include/net/netns/ipv4.h
@@ -26,6 +26,8 @@ struct netns_ipv4 {
  struct hlist_head *fib_table_hash;
  struct sock  *fibnl;

+ struct sock  **icmp_sk;
+
  struct netns_frags frags;
 #ifdef CONFIG_NETFILTER
  struct xt_table  *iptable_filter;
diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index 97d97ad..b51f4b0 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -229,8 +229,10 @@ static const struct icmp_control icmp_pointers[NR_ICMP_TYPES+1];
  *
  * On SMP we have one ICMP socket per-cpu.
  */
-static struct sock **__icmp_sk = NULL;
-#define icmp_sk  (__icmp_sk[smp_processor_id()])
+static struct sock *icmp_sk(struct net *net)
+{
+ return net->ipv4.icmp_sk[smp_processor_id()];
+}

 static inline int icmp_xmit_lock(struct sock *sk)
 {
@@ -349,7 +351,7 @@ static void icmp_push_reply(struct icmp_bxm *icmp_param,
  struct sock *sk;
  struct sk_buff *skb;

- sk = icmp_sk;
+ sk = icmp_sk(rt->u.dst.dev->nd_net);
  if (ip_append_data(sk, icmp_glue_bits, icmp_param,
       icmp_param->data_len+icmp_param->head_len,
       icmp_param->head_len,
@@ -378,10 +380,11 @@ static void icmp_push_reply(struct icmp_bxm *icmp_param,

 static void icmp_reply(struct icmp_bxm *icmp_param, struct sk_buff *skb)
 {
- struct sock *sk = icmp_sk;
- struct inet_sock *inet = inet_sk(sk);
  struct ipcm_cookie ipc;
  struct rtable *rt = (struct rtable *)skb->dst;
+ struct net *net = rt->u.dst.dev->nd_net;
```

```
+ struct sock *sk = icmp_sk(net);
+ struct inet_sock *inet = inet_sk(sk);
  __be32 daddr;

  if (ip_options_echo(&icmp_param->replyopts, skb))
@@ -407,7 +410,7 @@ static void icmp_reply(struct icmp_bxm *icmp_param, struct sk_buff
*skb)
       .tos = RT_TOS(ip_hdr(skb)->tos) } },
         .proto = IPPROTO_ICMP };
   security_skb_classify_flow(skb, &fl);
-  if (ip_route_output_key(rt->u.dst.dev->nd_net, &rt, &fl))
+  if (ip_route_output_key(net, &rt, &fl))
    goto out_unlock;
  }
  if (icmpv4_xrlim_allow(rt, icmp_param->data.icmph.type,
@@ -440,11 +443,12 @@ void icmp_send(struct sk_buff *skb_in, int type, int code, __be32 info)
  __be32 saddr;
  u8  tos;
  struct net *net;
- struct sock *sk = icmp_sk;
+ struct sock *sk;

  if (!rt)
   goto out;
  net = rt->u.dst.dev->nd_net;
+ sk = icmp_sk(net);

  /*
   * Find the original header. It is expected to be valid, of course.
@@ -1142,22 +1146,23 @@ static const struct icmp_control icmp_pointers[NR_ICMP_TYPES +
1] = {
  },
 };

-static void __exit icmp_exit(void)
+static void __net_exit icmp_sk_exit(struct net *net)
 {
  int i;

  for_each_possible_cpu(i)
-  sk_release_kernel(__icmp_sk[i]);
- kfree(__icmp_sk);
- __icmp_sk = NULL;
+  sk_release_kernel(net->ipv4.icmp_sk[i]);
+ kfree(net->ipv4.icmp_sk);
+ net->ipv4.icmp_sk = NULL;
 }
```

```
-int __init icmp_init(void)
+int __net_init icmp_sk_init(struct net *net)
 {
  int i, err;

- __icmp_sk = kzalloc(nr_cpu_ids * sizeof(struct sock *), GFP_KERNEL);
- if (__icmp_sk == NULL)
+ net->ipv4.icmp_sk =
+  kzalloc(nr_cpu_ids * sizeof(struct sock *), GFP_KERNEL);
+ if (net->ipv4.icmp_sk == NULL)
   return -ENOMEM;

  for_each_possible_cpu(i) {
@@ -1169,8 +1174,8 @@ int __init icmp_init(void)
   if (err < 0)
    goto fail;

-  __icmp_sk[i] = sk = sock->sk;
-  sk_change_net(sk, &init_net);
+  net->ipv4.icmp_sk[i] = sk = sock->sk;
+  sk_change_net(sk, net);

   sk->sk_allocation = GFP_ATOMIC;

@@ -1193,10 +1198,20 @@ int __init icmp_init(void)
  return 0;

 fail:
- icmp_exit();
+ icmp_sk_exit(net);
  return err;
 }

+static struct pernet_operations __net_initdata icmp_sk_ops = {
+       .init = icmp_sk_init,
+       .exit = icmp_sk_exit,
+};
+
+int __init icmp_init(void)
+{
+ return register_pernet_device(&icmp_sk_ops);
+}
+
 EXPORT_SYMBOL(icmp_err_convert);
 EXPORT_SYMBOL(icmp_send);
 EXPORT_SYMBOL(icmp_statistics);
--
1.5.3.rc5
```

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

Subject: [PATCH 11/12] [NETNS]: Make icmpv6_sk per namespace.
Posted by den on Fri, 29 Feb 2008 13:40:57 GMT
View Forum Message <> Reply to Message

All preparations are done. Now just add a hook to perform an initialization
on namespace startup and replace icmpv6_sk macro with proper inline call.
Actual namespace the packet belongs too will be passed later along with the
one for the routing.

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>

```
---
 include/net/netns/ipv6.h |    1 +
 net/ipv6/icmp.c          |   68 +++++++++++++++++++++++++++++++--------------
 2 files changed, 48 insertions(+), 21 deletions(-)

diff --git a/include/net/netns/ipv6.h b/include/net/netns/ipv6.h
index 1dd7de4..82623d3 100644
--- a/include/net/netns/ipv6.h
+++ b/include/net/netns/ipv6.h
@@ -36,5 +36,6 @@ struct netns_ipv6 {
  struct xt_table  *ip6table_mangle;
  struct xt_table  *ip6table_raw;
 #endif
+ struct sock  **icmp_sk;
 };
 #endif
diff --git a/net/ipv6/icmp.c b/net/ipv6/icmp.c
index 7341d79..9f55a96 100644
--- a/net/ipv6/icmp.c
+++ b/net/ipv6/icmp.c
@@ -80,8 +80,10 @@ EXPORT_SYMBOL(icmpv6msg_statistics);
  *
  * On SMP we have one ICMP socket per-cpu.
  */
-static struct sock **__icmpv6_sk = NULL;
-#define icmpv6_sk (__icmpv6_sk[smp_processor_id()])
+static inline struct sock *icmpv6_sk(struct net *net)
+{
+ return net->ipv6.icmp_sk[smp_processor_id()];
+}
```

```
 static int icmpv6_rcv(struct sk_buff *skb);

@@ -389,7 +391,7 @@ void icmpv6_send(struct sk_buff *skb, int type, int code, __u32 info,
  fl.fl_icmp_code = code;
  security_skb_classify_flow(skb, &fl);

- sk = icmpv6_sk;
+ sk = icmpv6_sk(&init_net);
  np = inet6_sk(sk);

  if (icmpv6_xmit_lock(sk))
@@ -535,7 +537,7 @@ static void icmpv6_echo_reply(struct sk_buff *skb)
  fl.fl_icmp_type = ICMPV6_ECHO_REPLY;
  security_skb_classify_flow(skb, &fl);

- sk = icmpv6_sk;
+ sk = icmpv6_sk(&init_net);
  np = inet6_sk(sk);

  if (icmpv6_xmit_lock(sk))
@@ -780,13 +782,14 @@ drop_no_count:
  */
 static struct lock_class_key icmpv6_socket_sk_dst_lock_key;

-int __init icmpv6_init(void)
+static int __net_init icmpv6_sk_init(struct net *net)
 {
  struct sock *sk;
  int err, i, j;

- __icmpv6_sk = kzalloc(nr_cpu_ids * sizeof(struct sock *), GFP_KERNEL);
- if (__icmpv6_sk == NULL)
+ net->ipv6.icmp_sk =
+  kzalloc(nr_cpu_ids * sizeof(struct sock *), GFP_KERNEL);
+ if (net->ipv6.icmp_sk == NULL)
   return -ENOMEM;

  for_each_possible_cpu(i) {
@@ -801,8 +804,8 @@ int __init icmpv6_init(void)
    goto fail;
   }

-  __icmpv6_sk[i] = sk = sock->sk;
-   sk_change_net(sk, &init_net);
+  net->ipv6.icmp_sk[i] = sk = sock->sk;
+  sk_change_net(sk, net);
```

```
   sk->sk_allocation = GFP_ATOMIC;
   /*
@@ -822,33 +825,56 @@ int __init icmpv6_init(void)

   sk->sk_prot->unhash(sk);
  }
-
-
- if (inet6_add_protocol(&icmpv6_protocol, IPPROTO_ICMPV6) < 0) {
-  printk(KERN_ERR "Failed to register ICMP6 protocol\n");
-  err = -EAGAIN;
-  goto fail;
- }
-
  return 0;

  fail:
  for (j = 0; j < i; j++)
-  sk_release_kernel(__icmpv6_sk[j]);
-
+  sk_release_kernel(net->ipv6.icmp_sk[j]);
+ kfree(net->ipv6.icmp_sk);
  return err;
 }

-void icmpv6_cleanup(void)
+static void __net_exit icmpv6_sk_exit(struct net *net)
 {
  int i;

  for_each_possible_cpu(i) {
-  sk_release_kernel(__icmpv6_sk[i]);
+  sk_release_kernel(net->ipv6.icmp_sk[i]);
  }
+ kfree(net->ipv6.icmp_sk);
+}
+
+static struct pernet_operations __net_initdata icmpv6_sk_ops = {
+       .init = icmpv6_sk_init,
+       .exit = icmpv6_sk_exit,
+};
+
+int __init icmpv6_init(void)
+{
+ int err;
+
+ err = register_pernet_subsys(&icmpv6_sk_ops);
+ if (err < 0)
```

```
+  return err;
+
+ err = -EAGAIN;
+ if (inet6_add_protocol(&icmpv6_protocol, IPPROTO_ICMPV6) < 0)
+  goto fail;
+ return 0;
+
+fail:
+ printk(KERN_ERR "Failed to register ICMP6 protocol\n");
+ unregister_pernet_subsys(&icmpv6_sk_ops);
+ return err;
+}
+
+void __exit icmpv6_cleanup(void)
+{
+ unregister_pernet_subsys(&icmpv6_sk_ops);
  inet6_del_protocol(&icmpv6_protocol, IPPROTO_ICMPV6);
 }

+
 static const struct icmp6_err {
  int err;
  int fatal;
--
1.5.3.rc5
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: [PATCH 12/12 net-2.6.26] [ICMP6]: Consolidate fail path icmpv6_sk_init with icmpv6_sk_exit.
Posted by den on Fri, 29 Feb 2008 13:40:58 GMT
View Forum Message <> Reply to Message

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>
---
 net/ipv6/icmp.c | 29 +++++++++++++++++----------
 1 files changed, 14 insertions(+), 15 deletions(-)

```
diff --git a/net/ipv6/icmp.c b/net/ipv6/icmp.c
index 9f55a96..889769d 100644
--- a/net/ipv6/icmp.c
+++ b/net/ipv6/icmp.c
@@ -777,6 +777,17 @@ drop_no_count:
```

```
  return 0;
 }

+
+static void __net_exit icmpv6_sk_exit(struct net *net)
+{
+ int i;
+
+ for_each_possible_cpu(i) {
+  sk_release_kernel(net->ipv6.icmp_sk[i]);
+ }
+ kfree(net->ipv6.icmp_sk);
+}
+
 /*
  * Special lock-class for __icmpv6_sk:
  */
@@ -785,7 +796,7 @@ static struct lock_class_key icmpv6_socket_sk_dst_lock_key;
 static int __net_init icmpv6_sk_init(struct net *net)
 {
  struct sock *sk;
- int err, i, j;
+ int err, i;

  net->ipv6.icmp_sk =
   kzalloc(nr_cpu_ids * sizeof(struct sock *), GFP_KERNEL);
@@ -827,23 +838,11 @@ static int __net_init icmpv6_sk_init(struct net *net)
  }
  return 0;

- fail:
- for (j = 0; j < i; j++)
-  sk_release_kernel(net->ipv6.icmp_sk[j]);
- kfree(net->ipv6.icmp_sk);
+fail:
+ icmpv6_sk_exit(net);
  return err;
 }

-static void __net_exit icmpv6_sk_exit(struct net *net)
-{
- int i;
-
- for_each_possible_cpu(i) {
-  sk_release_kernel(net->ipv6.icmp_sk[i]);
- }
- kfree(net->ipv6.icmp_sk);
-}
```

```
-
 static struct pernet_operations __net_initdata icmpv6_sk_ops = {
        .init = icmpv6_sk_init,
        .exit = icmpv6_sk_exit,
--
1.5.3.rc5
```

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 0/12 net-2.6.26] icmp_socket namespacing
Posted by davem on Fri, 29 Feb 2008 19:23:11 GMT
View Forum Message <> Reply to Message

From: "Denis V. Lunev" <den@openvz.org>
Date: Fri, 29 Feb 2008 16:39:20 +0300

> This set of patches is targeted to create separate icmp_socket inside
> each namespace. Both IPv4 and IPv6 codepaths are affected.
>
> Though, in order to do this smoothly, a bit of optimisations are
> performed. The kernel from now on will use sock rather than socket on
> the ICMP send path.
>
> Signed-off-by: Denis V. Lunev <den@openvz.org>

I applied everything except the final patch #12.

I'll reply to that patch posting with my feedback.

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 12/12 net-2.6.26] [ICMP6]: Consolidate fail path icmpv6_sk_init with icmpv6_sk_exit.
Posted by davem on Fri, 29 Feb 2008 19:23:42 GMT
View Forum Message <> Reply to Message

From: "Denis V. Lunev" <den@openvz.org>
Date: Fri, 29 Feb 2008 16:40:58 +0300

> +static void __net_exit icmpv6_sk_exit(struct net *net)

> +{

Since you call this from __new_init code, won't this
create a section conflict when ipv6 is built statically
into the kernel?

_____
Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: [PATCH 9/12 net-2.6.26] [NETNS]: icmp(v6)_sk should not pin a namespace.
Posted by davem on Fri, 29 Feb 2008 19:35:25 GMT
View Forum Message <> Reply to Message

From: "Denis V. Lunev" <den@openvz.org>
Date: Fri, 29 Feb 2008 16:40:55 +0300

> So, change icmp(v6)_sk creation/disposal to the scheme used in the netlink
> for rtnl, i.e. create a socket in the context of the init_net and
> assign the namespace without getting a referrence later.
>
> Also use sk_release_kernel instead of sock_release to properly destroy
> such sockets.
>
> Signed-off-by: Denis V. Lunev <den@openvz.org>
> Acked-by: Daniel Lezcano <dlezcano@fr.ibm.com>

Please validate the build in the future, thanks :-/

I just checked in the following:

commit 45af1754bc09926b5e062bda24f789d7b320939f
Author: David S. Miller <davem@davemloft.net>
Date: Fri Feb 29 11:33:19 2008 -0800

    [NET]: sk_release_kernel needs to be exported to modules

    Fixes:

    ERROR: "sk_release_kernel" [net/ipv6/ipv6.ko] undefined!

    Signed-off-by: David S. Miller <davem@davemloft.net>

diff --git a/net/core/sock.c b/net/core/sock.c
index c71b645..0ca0697 100644
--- a/net/core/sock.c

```
+++ b/net/core/sock.c
@@ -1004,6 +1004,7 @@ void sk_release_kernel(struct sock *sk)
  sk->sk_net = get_net(&init_net);
  sock_put(sk);
 }
+EXPORT_SYMBOL(sk_release_kernel);

 struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
 {
```

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Purpose of __net_exit & friends [Was: [ICMP6]: Consolidate fail ...]
Posted by Sam Ravnborg on Fri, 29 Feb 2008 20:12:47 GMT
View Forum Message <> Reply to Message

On Fri, Feb 29, 2008 at 11:23:42AM -0800, David Miller wrote:
> From: "Denis V. Lunev" <den@openvz.org>
> Date: Fri, 29 Feb 2008 16:40:58 +0300
>
> > +static void __net_exit icmpv6_sk_exit(struct net *net)
> > +{
>
> Since you call this from __new_init code, won't this
> create a section conflict when ipv6 is built statically
> into the kernel?

I have noticed this __net_init/__net_exit stuff before
but never got around to ask about it.

Whats the actual purpose and are there any strict
rules as to what may be called from where?
If there are we should apply the same checks as we do
for __devinit/__devexit and friends.

 Sam

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers

---

Subject: Re: Purpose of __net_exit & friends
Posted by davem on Fri, 29 Feb 2008 20:23:55 GMT

From: Sam Ravnborg <sam@ravnborg.org>
Date: Fri, 29 Feb 2008 21:12:47 +0100

> On Fri, Feb 29, 2008 at 11:23:42AM -0800, David Miller wrote:
> > From: "Denis V. Lunev" <den@openvz.org>
> > Date: Fri, 29 Feb 2008 16:40:58 +0300
> >
> > > +static void __net_exit icmpv6_sk_exit(struct net *net)
> > > +{
> >
> > Since you call this from __new_init code, won't this
> > create a section conflict when ipv6 is built statically
> > into the kernel?
>
> I have noticed this __net_init/__net_exit stuff before
> but never got around to ask about it.
>
> Whats the actual purpose and are there any strict
> rules as to what may be called from where?
> If there are we should apply the same checks as we do
> for __devinit/__devexit and friends.

Basically, if network namespace support isn't enabled,
these functions can act like normal __init and __exit
functions.

Otherwise we have to keep them around so that they can
be called when namespaces are created and destroyed.

Subject: Re: [PATCH 12/12 net-2.6.26] [ICMP6]: Consolidate fail path icmpv6_sk_init with icmpv6_sk_exit.
Posted by den on Fri, 29 Feb 2008 22:05:41 GMT

On Fri, 2008-02-29 at 11:23 -0800, David Miller wrote:
> From: "Denis V. Lunev" <den@openvz.org>
> Date: Fri, 29 Feb 2008 16:40:58 +0300
>
> > +static void __net_exit icmpv6_sk_exit(struct net *net)
> > +{
>

> Since you call this from __new_init code, won't this
> create a section conflict when ipv6 is built statically
> into the kernel?

Dave, you are perfectly correct :) Though, I have made a similar mistake
in the IPv4 code. Pls consider the patch attached.

[ICMP]: Section conflict between icmp_sk_init/icmp_sk_exit.

Functions from __exit section should not be called from ones in __init
section. Fix this conflict.

Signed-off-by: Denis V. Lunev <den@openvz.org>

```
diff --git a/net/ipv4/icmp.c b/net/ipv4/icmp.c
index b51f4b0..cee77d6 100644
--- a/net/ipv4/icmp.c
+++ b/net/ipv4/icmp.c
@@ -1198,7 +1198,9 @@ int __net_init icmp_sk_init(struct net *net)
  return 0;

 fail:
- icmp_sk_exit(net);
+ for_each_possible_cpu(i)
+  sk_release_kernel(net->ipv4.icmp_sk[i]);
+ kfree(net->ipv4.icmp_sk);
  return err;
 }
```


File Attachments

1) diff-icmp-section.txt, downloaded 356 times

Subject: Re: [PATCH 12/12 net-2.6.26] [ICMP6]: Consolidate fail path icmpv6_sk_init with icmpv6_sk_exit.
Posted by davem on Fri, 29 Feb 2008 22:15:45 GMT
View Forum Message <> Reply to Message

From: "Denis V. Lunev" <den@openvz.org>
Date: Sat, 01 Mar 2008 01:05:41 +0300

> Though, I have made a similar mistake in the IPv4 code. Pls consider

> the patch attached.

Applied and pushed out to net-2.6.26, thanks.

_____

Containers mailing list
Containers@lists.linux-foundation.org
https://lists.linux-foundation.org/mailman/listinfo/containers