Subject: [PATCH 4/9] network namespaces: socket hashes
Posted by Andrey Savochkin on Tue, 15 Aug 2006 14:48:44 GMT
View Forum Message <> Reply to Message

Socket hash lookups are made within namespace.
Hash tables are common for all namespaces, with
additional permutation of indexes.

Signed-off-by: Andrey Savochkin <saw@swsoft.com>
---
 include/linux/ipv6.h | 3 ++-
 include/net/inet6_hashtables.h | 6 ++++--
 include/net/inet_hashtables.h | 38 ++++++++++++++++++++++++++++++++------
 include/net/inet_sock.h | 6 ++++--
 include/net/inet_timewait_sock.h | 2 ++
 include/net/sock.h | 4 ++++
 include/net/udp.h | 12 ++++++++++---
 net/core/sock.c | 5 +++++
 net/ipv4/inet_connection_sock.c | 19 +++++++++++++++----
 net/ipv4/inet_hashtables.c | 29 +++++++++++++++++++++++------
 net/ipv4/inet_timewait_sock.c | 8 ++++++--
 net/ipv4/raw.c | 2 ++
 net/ipv4/udp.c | 20 ++++++++++++++++----
 net/ipv6/inet6_connection_sock.c | 2 ++
 net/ipv6/inet6_hashtables.c | 25 +++++++++++++++++++------
 net/ipv6/raw.c | 4 ++++
 net/ipv6/udp.c | 21 ++++++++++++++++-----
 17 files changed, 151 insertions(+), 55 deletions(-)

--- ./include/linux/ipv6.h.venssock Mon Aug 14 17:02:45 2006
+++ ./include/linux/ipv6.h Tue Aug 15 13:38:47 2006
@@ -428,10 +428,11 @@ static inline struct raw6_sock *raw6_sk(
 #define inet_v6_ipv6only(__sk) 0
 #endif /* defined(CONFIG_IPV6) || defined(CONFIG_IPV6_MODULE) */

-#define INET6_MATCH(__sk, __hash, __saddr, __daddr, __ports, __dif)\
+#define INET6_MATCH(__sk, __hash, __saddr, __daddr, __ports, __dif, __ns)\
   (((__sk)->sk_hash == (__hash))   && \
    ((*((__u32 *)&(inet_sk(__sk)->dport))) == (__ports))   && \
    ((__sk)->sk_family == AF_INET6) && \
+ net_ns_match((__sk)->sk_net_ns, __ns)   && \
    ipv6_addr_equal(&inet6_sk(__sk)->daddr, (__saddr)) && \
    ipv6_addr_equal(&inet6_sk(__sk)->rcv_saddr, (__daddr)) && \
    (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
--- ./include/net/inet6_hashtables.h.venssock Mon Aug 14 17:02:47 2006
+++ ./include/net/inet6_hashtables.h Tue Aug 15 13:38:47 2006
@@ -26,11 +26,13 @@ struct inet_hashinfo;

```
 /* I have no idea if this is a good hash for v6 or not. -DaveM */
 static inline unsigned int inet6_ehashfn(const struct in6_addr *laddr, const u16 lport,
-    const struct in6_addr *faddr, const u16 fport)
+    const struct in6_addr *faddr, const u16 fport,
+    struct net_namespace *ns)
 {
  unsigned int hashent = (lport ^ fport);

  hashent ^= (laddr->s6_addr32[3] ^ faddr->s6_addr32[3]);
+ hashent ^= net_ns_hash(ns);
  hashent ^= hashent >> 16;
  hashent ^= hashent >> 8;
  return hashent;
@@ -44,7 +46,7 @@ static inline int inet6_sk_ehashfn(const
  const struct in6_addr *faddr = &np->daddr;
  const __u16 lport = inet->num;
  const __u16 fport = inet->dport;
- return inet6_ehashfn(laddr, lport, faddr, fport);
+ return inet6_ehashfn(laddr, lport, faddr, fport, current_net_ns);
 }

 extern void __inet6_hash(struct inet_hashinfo *hashinfo, struct sock *sk);
--- ./include/net/inet_hashtables.h.venssock Mon Aug 14 17:04:04 2006
+++ ./include/net/inet_hashtables.h Tue Aug 15 13:38:47 2006
@@ -74,6 +74,9 @@ struct inet_ehash_bucket {
  * ports are created in O(1) time?  I thought so. ;-) -DaveM
  */
 struct inet_bind_bucket {
+#ifdef CONFIG_NET_NS
+ struct net_namespace *net_ns;
+#endif
  unsigned short  port;
  signed short  fastreuse;
  struct hlist_node node;
@@ -142,30 +145,34 @@ extern struct inet_bind_bucket *
 extern void inet_bind_bucket_destroy(kmem_cache_t *cachep,
         struct inet_bind_bucket *tb);

-static inline int inet_bhashfn(const __u16 lport, const int bhash_size)
+static inline int inet_bhashfn(const __u16 lport,
+         struct net_namespace *ns,
+         const int bhash_size)
 {
- return lport & (bhash_size - 1);
+ return (lport ^ net_ns_hash(ns)) & (bhash_size - 1);
 }

 extern void inet_bind_hash(struct sock *sk, struct inet_bind_bucket *tb,
```

```
       const unsigned short snum);

 /* These can have wildcards, don't try too hard. */
-static inline int inet_lhashfn(const unsigned short num)
+static inline int inet_lhashfn(const unsigned short num,
+          struct net_namespace *ns)
 {
- return num & (INET_LHTABLE_SIZE - 1);
+ return (num ^ net_ns_hash(ns)) & (INET_LHTABLE_SIZE - 1);
 }

 static inline int inet_sk_listen_hashfn(const struct sock *sk)
 {
- return inet_lhashfn(inet_sk(sk)->num);
+ return inet_lhashfn(inet_sk(sk)->num, current_net_ns);
 }

 /* Caller must disable local BH processing. */
 static inline void __inet_inherit_port(struct inet_hashinfo *table,
          struct sock *sk, struct sock *child)
 {
- const int bhash = inet_bhashfn(inet_sk(child)->num, table->bhash_size);
+ const int bhash = inet_bhashfn(inet_sk(child)->num, current_net_ns,
+     table->bhash_size);
  struct inet_bind_hashbucket *head = &table->bhash[bhash];
  struct inet_bind_bucket *tb;

@@ -299,29 +306,33 @@ static inline struct sock *inet_lookup_l
 #define INET_ADDR_COOKIE(__name, __saddr, __daddr) \
  const __u64 __name = (((__u64)(__daddr)) << 32) | ((__u64)(__saddr));
 #endif /* __BIG_ENDIAN */
-#define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif)\
+#define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif, __ns)\
   (((__sk)->sk_hash == (__hash))    && \
    ((*((__u64 *)&(inet_sk(__sk)->daddr))) == (__cookie)) && \
    ((*((__u32 *)&(inet_sk(__sk)->dport))) == (__ports)) && \
+  net_ns_match((__sk)->sk_net_ns, __ns)   && \
    (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
-#define INET_TW_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif)\
+#define INET_TW_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif, __ns)\
   (((__sk)->sk_hash == (__hash))    && \
    ((*((__u64 *)&(inet_twsk(__sk)->tw_daddr))) == (__cookie)) && \
    ((*((__u32 *)&(inet_twsk(__sk)->tw_dport))) == (__ports)) && \
+  net_ns_match((__sk)->sk_net_ns, __ns)   && \
    (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
 #else /* 32-bit arch */
 #define INET_ADDR_COOKIE(__name, __saddr, __daddr)
-#define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif) \
```

```
+#define INET_MATCH(__sk, __hash, __cookie, __saddr, __daddr, __ports, __dif, __ns)\
  (((__sk)->sk_hash == (__hash))    && \
   (inet_sk(__sk)->daddr  == (__saddr))  && \
   (inet_sk(__sk)->rcv_saddr == (__daddr))  && \
   ((*((__u32 *)&(inet_sk(__sk)->dport))) == (__ports)) && \
+  net_ns_match((__sk)->sk_net_ns, __ns)   && \
   (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
-#define INET_TW_MATCH(__sk, __hash,__cookie, __saddr, __daddr, __ports, __dif) \
+#define INET_TW_MATCH(__sk, __hash,__cookie, __saddr, __daddr, __ports, __dif, __ns)\
  (((__sk)->sk_hash == (__hash))    && \
   (inet_twsk(__sk)->tw_daddr == (__saddr))  && \
   (inet_twsk(__sk)->tw_rcv_saddr == (__daddr))  && \
   ((*((__u32 *)&(inet_twsk(__sk)->tw_dport))) == (__ports)) && \
+  net_ns_match((__sk)->sk_net_ns, __ns)   && \
   (!((__sk)->sk_bound_dev_if) || ((__sk)->sk_bound_dev_if == (__dif))))
 #endif /* 64-bit arch */

@@ -341,22 +352,23 @@ static inline struct sock *
  const __u32 ports = INET_COMBINED_PORTS(sport, hnum);
  struct sock *sk;
  const struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;
  /* Optimize here for direct hit, only listening connections can
   * have wildcards anyways.
   */
- unsigned int hash = inet_ehashfn(daddr, hnum, saddr, sport);
+ unsigned int hash = inet_ehashfn(daddr, hnum, saddr, sport, ns);
  struct inet_ehash_bucket *head = inet_ehash_bucket(hashinfo, hash);

  prefetch(head->chain.first);
  read_lock(&head->lock);
  sk_for_each(sk, node, &head->chain) {
-  if (INET_MATCH(sk, hash, acookie, saddr, daddr, ports, dif))
+  if (INET_MATCH(sk, hash, acookie, saddr, daddr, ports, dif, ns))
    goto hit; /* You sunk my battleship! */
  }

  /* Must check for a TIME_WAIT'er before going to listener hash. */
  sk_for_each(sk, node, &(head + hashinfo->ehash_size)->chain) {
-  if (INET_TW_MATCH(sk, hash, acookie, saddr, daddr, ports, dif))
+  if (INET_TW_MATCH(sk, hash, acookie, saddr, daddr, ports, dif, ns))
    goto hit;
  }
  sk = NULL;
--- ./include/net/inet_sock.h.venssock Mon Aug 14 17:04:04 2006
+++ ./include/net/inet_sock.h Tue Aug 15 13:38:47 2006
@@ -168,9 +168,11 @@ static inline void inet_sk_copy_descenda
 extern int inet_sk_rebuild_header(struct sock *sk);
```

```
 static inline unsigned int inet_ehashfn(const __u32 laddr, const __u16 lport,
-     const __u32 faddr, const __u16 fport)
+     const __u32 faddr, const __u16 fport,
+     struct net_namespace *ns)
 {
  unsigned int h = (laddr ^ lport) ^ (faddr ^ fport);
+ h ^= net_ns_hash(ns);
  h ^= h >> 16;
  h ^= h >> 8;
  return h;
@@ -184,7 +186,7 @@ static inline int inet_sk_ehashfn(const
  const __u32 faddr = inet->daddr;
  const __u16 fport = inet->dport;

- return inet_ehashfn(laddr, lport, faddr, fport);
+ return inet_ehashfn(laddr, lport, faddr, fport, current_net_ns);
 }

 #endif /* _INET_SOCK_H */
--- ./include/net/inet_timewait_sock.h.venssock Mon Aug 14 17:02:47 2006
+++ ./include/net/inet_timewait_sock.h Tue Aug 15 13:38:47 2006
@@ -115,6 +115,7 @@ struct inet_timewait_sock {
 #define tw_refcnt  __tw_common.skc_refcnt
 #define tw_hash    __tw_common.skc_hash
 #define tw_prot    __tw_common.skc_prot
+#define tw_net_ns  __tw_common.skc_net_ns
  volatile unsigned char tw_substate;
  /* 3 bits hole, try to pack */
  unsigned char  tw_rcv_wscale;
@@ -200,6 +201,7 @@ static inline void inet_twsk_put(struct
   printk(KERN_DEBUG "%s timewait_sock %p released\n",
          tw->tw_prot->name, tw);
 #endif
+  put_net_ns(tw->tw_net_ns);
   kmem_cache_free(tw->tw_prot->twsk_prot->twsk_slab, tw);
   module_put(owner);
  }
--- ./include/net/sock.h.venssock Mon Aug 14 17:04:05 2006
+++ ./include/net/sock.h Tue Aug 15 13:38:47 2006
@@ -118,6 +118,9 @@ struct sock_common {
  atomic_t  skc_refcnt;
  unsigned int  skc_hash;
  struct proto  *skc_prot;
+#ifdef CONFIG_NET_NS
+ struct net_namespace *skc_net_ns;
+#endif
 };
```

```
 /**
@@ -194,6 +197,7 @@ struct sock {
 #define sk_refcnt  __sk_common.skc_refcnt
 #define sk_hash   __sk_common.skc_hash
 #define sk_prot   __sk_common.skc_prot
+#define sk_net_ns  __sk_common.skc_net_ns
  unsigned char  sk_shutdown : 2,
     sk_no_check : 2,
     sk_userlocks : 4;
--- ./include/net/udp.h.venssock Mon Mar 20 08:53:29 2006
+++ ./include/net/udp.h Tue Aug 15 13:38:47 2006
@@ -39,13 +39,19 @@ extern rwlock_t udp_hash_lock;

 extern int udp_port_rover;

-static inline int udp_lport_inuse(u16 num)
+static inline int udp_hashfn(u16 num, struct net_namespace *ns)
+{
+ return (num ^ net_ns_hash(ns)) & (UDP_HTABLE_SIZE - 1);
+}
+
+static inline int udp_lport_inuse(u16 num, struct net_namespace *ns)
 {
  struct sock *sk;
  struct hlist_node *node;

- sk_for_each(sk, node, &udp_hash[num & (UDP_HTABLE_SIZE - 1)])
-  if (inet_sk(sk)->num == num)
+ sk_for_each(sk, node, &udp_hash[udp_hashfn(num, ns)])
+  if (inet_sk(sk)->num == num &&
+      net_ns_match(sk->sk_net_ns, ns))
    return 1;
  return 0;
 }
--- ./net/core/sock.c.venssock Mon Aug 14 17:04:05 2006
+++ ./net/core/sock.c Tue Aug 15 13:38:47 2006
@@ -858,6 +858,9 @@ struct sock *sk_alloc(int family, gfp_t
     */
    sk->sk_prot = sk->sk_prot_creator = prot;
    sock_lock_init(sk);
+#ifdef CONFIG_NET_NS
+   sk->sk_net_ns = get_net_ns(current_net_ns);
+#endif
   }

   if (security_sk_alloc(sk, family, priority))
@@ -897,6 +900,7 @@ void sk_free(struct sock *sk)
```

```
        __FUNCTION__, atomic_read(&sk->sk_omem_alloc));

  security_sk_free(sk);
+ put_net_ns(sk->sk_net_ns);
  if (sk->sk_prot_creator->slab != NULL)
   kmem_cache_free(sk->sk_prot_creator->slab, sk);
  else
@@ -932,6 +936,7 @@ struct sock *sk_clone(const struct sock
   lockdep_set_class(&newsk->sk_callback_lock,
        af_callback_keys + newsk->sk_family);

+  (void) get_net_ns(newsk->sk_net_ns);
   newsk->sk_dst_cache = NULL;
   newsk->sk_wmem_queued = 0;
   newsk->sk_forward_alloc = 0;
--- ./net/ipv4/inet_connection_sock.c.venssock Mon Aug 14 17:04:08 2006
+++ ./net/ipv4/inet_connection_sock.c Tue Aug 15 13:38:47 2006
@@ -43,10 +43,12 @@ int inet_csk_bind_conflict(const struct
  struct sock *sk2;
  struct hlist_node *node;
  int reuse = sk->sk_reuse;
+ struct net_namespace *ns = current_net_ns;

  sk_for_each_bound(sk2, node, &tb->owners) {
   if (sk != sk2 &&
       !inet_v6_ipv6only(sk2) &&
+      net_ns_match(sk2->sk_net_ns, ns) &&
       (!sk->sk_bound_dev_if ||
        !sk2->sk_bound_dev_if ||
        sk->sk_bound_dev_if == sk2->sk_bound_dev_if)) {
@@ -75,6 +77,7 @@ int inet_csk_get_port(struct inet_hashin
  struct inet_bind_hashbucket *head;
  struct hlist_node *node;
  struct inet_bind_bucket *tb;
+ struct net_namespace *ns = current_net_ns;
  int ret;

  local_bh_disable();
@@ -85,11 +88,15 @@ int inet_csk_get_port(struct inet_hashin
   int rover = net_random() % (high - low) + low;

   do {
-   head = &hashinfo->bhash[inet_bhashfn(rover, hashinfo->bhash_size)];
+   head = &hashinfo->bhash[inet_bhashfn(rover, ns,
+       hashinfo->bhash_size)];
    spin_lock(&head->lock);
-   inet_bind_bucket_for_each(tb, node, &head->chain)
+   inet_bind_bucket_for_each(tb, node, &head->chain) {
```

```
+    if (!net_ns_match(tb->net_ns, ns))
+     continue;
     if (tb->port == rover)
      goto next;
+   }
    break;
   next:
    spin_unlock(&head->lock);
@@ -112,11 +119,15 @@ int inet_csk_get_port(struct inet_hashin
    */
   snum = rover;
  } else {
-  head = &hashinfo->bhash[inet_bhashfn(snum, hashinfo->bhash_size)];
+  head = &hashinfo->bhash[inet_bhashfn(snum, ns,
+       hashinfo->bhash_size)];
   spin_lock(&head->lock);
-  inet_bind_bucket_for_each(tb, node, &head->chain)
+  inet_bind_bucket_for_each(tb, node, &head->chain) {
+   if (!net_ns_match(tb->net_ns, ns))
+    continue;
    if (tb->port == snum)
     goto tb_found;
+  }
  }
  tb = NULL;
  goto tb_not_found;
--- ./net/ipv4/inet_hashtables.c.venssock Mon Aug 14 17:04:08 2006
+++ ./net/ipv4/inet_hashtables.c Tue Aug 15 13:43:20 2006
@@ -36,6 +36,9 @@ struct inet_bind_bucket *inet_bind_bucke
  if (tb != NULL) {
   tb->port      = snum;
   tb->fastreuse = 0;
+#ifdef CONFIG_NET_NS
+  tb->net_ns = get_net_ns(current_net_ns);
+#endif
   INIT_HLIST_HEAD(&tb->owners);
   hlist_add_head(&tb->node, &head->chain);
  }
@@ -49,6 +52,7 @@ void inet_bind_bucket_destroy(kmem_cache
 {
  if (hlist_empty(&tb->owners)) {
   __hlist_del(&tb->node);
+  put_net_ns(tb->net_ns);
   kmem_cache_free(cachep, tb);
  }
 }
@@ -66,7 +70,8 @@ void inet_bind_hash(struct sock *sk, str
  */
```

```
 static void __inet_put_port(struct inet_hashinfo *hashinfo, struct sock *sk)
 {
- const int bhash = inet_bhashfn(inet_sk(sk)->num, hashinfo->bhash_size);
+ const int bhash = inet_bhashfn(inet_sk(sk)->num, current_net_ns,
+     hashinfo->bhash_size);
  struct inet_bind_hashbucket *head = &hashinfo->bhash[bhash];
  struct inet_bind_bucket *tb;

@@ -130,12 +135,15 @@ static struct sock *inet_lookup_listener
           const int dif)
 {
  struct sock *result = NULL, *sk;
+ struct net_namespace *ns = current_net_ns;
  const struct hlist_node *node;
  int hiscore = -1;

  sk_for_each(sk, node, head) {
   const struct inet_sock *inet = inet_sk(sk);

+  if (!net_ns_match(sk->sk_net_ns, ns))
+   continue;
   if (inet->num == hnum && !ipv6_only_sock(sk)) {
    const __u32 rcv_saddr = inet->rcv_saddr;
    int score = sk->sk_family == PF_INET ? 1 : 0;
@@ -168,14 +176,16 @@ struct sock *__inet_lookup_listener(stru
 {
  struct sock *sk = NULL;
  const struct hlist_head *head;
+ struct net_namespace *ns = current_net_ns;

  read_lock(&hashinfo->lhash_lock);
- head = &hashinfo->listening_hash[inet_lhashfn(hnum)];
+ head = &hashinfo->listening_hash[inet_lhashfn(hnum, ns)];
  if (!hlist_empty(head)) {
   const struct inet_sock *inet = inet_sk((sk = __sk_head(head)));

   if (inet->num == hnum && !sk->sk_node.next &&
       (!inet->rcv_saddr || inet->rcv_saddr == daddr) &&
+      net_ns_match(sk->sk_net_ns, ns) &&
       (sk->sk_family == PF_INET || !ipv6_only_sock(sk)) &&
       !sk->sk_bound_dev_if)
    goto sherry_cache;
@@ -202,7 +212,8 @@ static int __inet_check_established(stru
  int dif = sk->sk_bound_dev_if;
  INET_ADDR_COOKIE(acookie, saddr, daddr)
  const __u32 ports = INET_COMBINED_PORTS(inet->dport, lport);
- unsigned int hash = inet_ehashfn(daddr, lport, saddr, inet->dport);
+ struct net_namespace *ns = current_net_ns;
```

```
+ unsigned int hash = inet_ehashfn(daddr, lport, saddr, inet->dport, ns);
  struct inet_ehash_bucket *head = inet_ehash_bucket(hinfo, hash);
  struct sock *sk2;
  const struct hlist_node *node;
@@ -215,7 +226,7 @@ static int __inet_check_established(stru
  sk_for_each(sk2, node, &(head + hinfo->ehash_size)->chain) {
   tw = inet_twsk(sk2);

-  if (INET_TW_MATCH(sk2, hash, acookie, saddr, daddr, ports, dif)) {
+  if (INET_TW_MATCH(sk2, hash, acookie, saddr, daddr, ports, dif, ns)) {
    if (twsk_unique(sk, sk2, twp))
     goto unique;
    else
@@ -226,7 +237,7 @@ static int __inet_check_established(stru

  /* And established part... */
  sk_for_each(sk2, node, &head->chain) {
-  if (INET_MATCH(sk2, hash, acookie, saddr, daddr, ports, dif))
+  if (INET_MATCH(sk2, hash, acookie, saddr, daddr, ports, dif, ns))
    goto not_unique;
  }

@@ -274,6 +285,7 @@ int inet_hash_connect(struct inet_timewa
 {
  struct inet_hashinfo *hinfo = death_row->hashinfo;
  const unsigned short snum = inet_sk(sk)->num;
+ struct net_namespace *ns = current_net_ns;
   struct inet_bind_hashbucket *head;
   struct inet_bind_bucket *tb;
  int ret;
@@ -292,7 +304,8 @@ int inet_hash_connect(struct inet_timewa
    local_bh_disable();
   for (i = 1; i <= range; i++) {
    port = low + (i + offset) % range;
-    head = &hinfo->bhash[inet_bhashfn(port, hinfo->bhash_size)];
+    head = &hinfo->bhash[inet_bhashfn(port, ns,
+      hinfo->bhash_size)];
     spin_lock(&head->lock);

     /* Does not bother with rcv_saddr checks,
@@ -300,6 +313,8 @@ int inet_hash_connect(struct inet_timewa
      * unique enough.
      */
    inet_bind_bucket_for_each(tb, node, &head->chain) {
+    if (!net_ns_match(tb->net_ns, ns))
+     continue;
      if (tb->port == port) {
       BUG_TRAP(!hlist_empty(&tb->owners));
```

```
       if (tb->fastreuse >= 0)
@@ -347,7 +362,7 @@ ok:
   goto out;
   }

-  head = &hinfo->bhash[inet_bhashfn(snum, hinfo->bhash_size)];
+  head = &hinfo->bhash[inet_bhashfn(snum, ns, hinfo->bhash_size)];
   tb  = inet_csk(sk)->icsk_bind_hash;
  spin_lock_bh(&head->lock);
  if (sk_head(&tb->owners) == sk && !sk->sk_bind_node.next) {
--- ./net/ipv4/inet_timewait_sock.c.venssock Mon Aug 14 17:02:49 2006
+++ ./net/ipv4/inet_timewait_sock.c Tue Aug 15 13:38:47 2006
@@ -31,7 +31,7 @@ void __inet_twsk_kill(struct inet_timewa
  write_unlock(&ehead->lock);

  /* Disassociate with bind bucket. */
- bhead = &hashinfo->bhash[inet_bhashfn(tw->tw_num, hashinfo->bhash_size)];
+ bhead = &hashinfo->bhash[inet_bhashfn(tw->tw_num, current_net_ns, hashinfo->bhash_size)];
  spin_lock(&bhead->lock);
  tb = tw->tw_tb;
  __hlist_del(&tw->tw_bind_node);
@@ -65,7 +65,8 @@ void __inet_twsk_hashdance(struct inet_t
     Note, that any socket with inet->num != 0 MUST be bound in
     binding cache, even if it is closed.
   */
- bhead = &hashinfo->bhash[inet_bhashfn(inet->num, hashinfo->bhash_size)];
+ bhead = &hashinfo->bhash[inet_bhashfn(inet->num, current_net_ns,
+     hashinfo->bhash_size)];
  spin_lock(&bhead->lock);
  tw->tw_tb = icsk->icsk_bind_hash;
  BUG_TRAP(icsk->icsk_bind_hash);
@@ -109,6 +110,9 @@ struct inet_timewait_sock *inet_twsk_all
   tw->tw_hash     = sk->sk_hash;
   tw->tw_ipv6only     = 0;
   tw->tw_prot     = sk->sk_prot_creator;
+#ifdef CONFIG_NET_NS
+  tw->tw_net_ns     = get_net_ns(current_net_ns);
+#endif
   atomic_set(&tw->tw_refcnt, 1);
   inet_twsk_dead_node_init(tw);
   __module_get(tw->tw_prot->owner);
--- ./net/ipv4/raw.c.venssock Mon Aug 14 17:04:10 2006
+++ ./net/ipv4/raw.c Tue Aug 15 13:38:47 2006
@@ -106,6 +106,7 @@ struct sock *__raw_v4_lookup(struct sock
         int dif)
 {
  struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;
```

```
  sk_for_each_from(sk, node) {
   struct inet_sock *inet = inet_sk(sk);
@@ -113,6 +114,7 @@ struct sock *__raw_v4_lookup(struct sock
   if (inet->num == num      &&
       !(inet->daddr && inet->daddr != raddr)   &&
       !(inet->rcv_saddr && inet->rcv_saddr != laddr) &&
+      net_ns_match(sk->sk_net_ns, ns)   &&
       !(sk->sk_bound_dev_if && sk->sk_bound_dev_if != dif))
    goto found; /* gotcha */
  }
--- ./net/ipv4/udp.c.venssock Mon Aug 14 17:04:10 2006
+++ ./net/ipv4/udp.c Tue Aug 15 13:38:47 2006
@@ -125,6 +125,7 @@ static int udp_v4_get_port(struct sock *
 {
  struct hlist_node *node;
  struct sock *sk2;
+ struct net_namespace *ns = current_net_ns;
  struct inet_sock *inet = inet_sk(sk);

  write_lock_bh(&udp_hash_lock);
@@ -140,7 +141,7 @@ static int udp_v4_get_port(struct sock *
    struct hlist_head *list;
    int size;

-   list = &udp_hash[result & (UDP_HTABLE_SIZE - 1)];
+   list = &udp_hash[udp_hashfn(result, ns)];
    if (hlist_empty(list)) {
     if (result > sysctl_local_port_range[1])
      result = sysctl_local_port_range[0] +
@@ -162,7 +163,7 @@ static int udp_v4_get_port(struct sock *
     result = sysctl_local_port_range[0]
      + ((result - sysctl_local_port_range[0]) &
        (UDP_HTABLE_SIZE - 1));
-   if (!udp_lport_inuse(result))
+   if (!udp_lport_inuse(result, ns))
     break;
   }
   if (i >= (1 << 16) / UDP_HTABLE_SIZE)
@@ -170,13 +171,13 @@ static int udp_v4_get_port(struct sock *
 gotit:
   udp_port_rover = snum = result;
  } else {
-  sk_for_each(sk2, node,
-      &udp_hash[snum & (UDP_HTABLE_SIZE - 1)]) {
+  sk_for_each(sk2, node, &udp_hash[udp_hashfn(snum, ns)]) {
    struct inet_sock *inet2 = inet_sk(sk2);
```

```
    if (inet2->num == snum &&
        sk2 != sk &&
        !ipv6_only_sock(sk2) &&
+       net_ns_match(sk2->sk_net_ns, ns) &&
        (!sk2->sk_bound_dev_if ||
         !sk->sk_bound_dev_if ||
         sk2->sk_bound_dev_if == sk->sk_bound_dev_if) &&
@@ -189,7 +190,7 @@ gotit:
  }
  inet->num = snum;
  if (sk_unhashed(sk)) {
-  struct hlist_head *h = &udp_hash[snum & (UDP_HTABLE_SIZE - 1)];
+  struct hlist_head *h = &udp_hash[udp_hashfn(snum, ns)];

   sk_add_node(sk, h);
   sock_prot_inc_use(sk->sk_prot);
@@ -225,12 +226,15 @@ static struct sock *udp_v4_lookup_longwa
 {
  struct sock *sk, *result = NULL;
  struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;
  unsigned short hnum = ntohs(dport);
  int badness = -1;

- sk_for_each(sk, node, &udp_hash[hnum & (UDP_HTABLE_SIZE - 1)]) {
+ sk_for_each(sk, node, &udp_hash[udp_hashfn(hnum, ns)]) {
   struct inet_sock *inet = inet_sk(sk);

+  if (!net_ns_match(sk->sk_net_ns, ns))
+   continue;
   if (inet->num == hnum && !ipv6_only_sock(sk)) {
    int score = (sk->sk_family == PF_INET ? 1 : 0);
    if (inet->rcv_saddr) {
@@ -285,6 +289,7 @@ static inline struct sock *udp_v4_mcast_
 {
  struct hlist_node *node;
  struct sock *s = sk;
+ struct net_namespace *ns = current_net_ns;
  unsigned short hnum = ntohs(loc_port);

  sk_for_each_from(s, node) {
@@ -295,6 +300,7 @@ static inline struct sock *udp_v4_mcast_
       (inet->dport != rmt_port && inet->dport)  ||
       (inet->rcv_saddr && inet->rcv_saddr != loc_addr) ||
       ipv6_only_sock(s)     ||
+      !net_ns_match(sk->sk_net_ns, ns)   ||
       (s->sk_bound_dev_if && s->sk_bound_dev_if != dif))
    continue;
```

```
   if (!ip_mc_sf_allow(s, loc_addr, rmt_addr, dif))
@@ -1063,7 +1069,7 @@ static int udp_v4_mcast_deliver(struct s
  int dif;

  read_lock(&udp_hash_lock);
- sk = sk_head(&udp_hash[ntohs(uh->dest) & (UDP_HTABLE_SIZE - 1)]);
+ sk = sk_head(&udp_hash[udp_hashfn(ntohs(uh->dest), current_net_ns)]);
  dif = skb->dev->ifindex;
  sk = udp_v4_mcast_next(sk, uh->dest, daddr, uh->source, saddr, dif);
  if (sk) {
--- ./net/ipv6/inet6_connection_sock.c.venssock Mon Aug 14 17:04:10 2006
+++ ./net/ipv6/inet6_connection_sock.c Tue Aug 15 13:38:47 2006
@@ -31,10 +31,12 @@ int inet6_csk_bind_conflict(const struct
 {
  const struct sock *sk2;
  const struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;

  /* We must walk the whole port owner list in this case. -DaveM */
  sk_for_each_bound(sk2, node, &tb->owners) {
   if (sk != sk2 &&
+      net_ns_match(sk2->sk_net_ns, ns) &&
       (!sk->sk_bound_dev_if ||
        !sk2->sk_bound_dev_if ||
        sk->sk_bound_dev_if == sk2->sk_bound_dev_if) &&
--- ./net/ipv6/inet6_hashtables.c.venssock Mon Aug 14 17:02:50 2006
+++ ./net/ipv6/inet6_hashtables.c Tue Aug 15 13:38:47 2006
@@ -65,17 +65,18 @@ struct sock *__inet6_lookup_established(
  struct sock *sk;
  const struct hlist_node *node;
  const __u32 ports = INET_COMBINED_PORTS(sport, hnum);
+ struct net_namespace *ns = current_net_ns;
  /* Optimize here for direct hit, only listening connections can
   * have wildcards anyways.
   */
- unsigned int hash = inet6_ehashfn(daddr, hnum, saddr, sport);
+ unsigned int hash = inet6_ehashfn(daddr, hnum, saddr, sport, ns);
  struct inet_ehash_bucket *head = inet_ehash_bucket(hashinfo, hash);

  prefetch(head->chain.first);
  read_lock(&head->lock);
  sk_for_each(sk, node, &head->chain) {
   /* For IPV6 do the cheaper port and family tests first. */
-  if (INET6_MATCH(sk, hash, saddr, daddr, ports, dif))
+  if (INET6_MATCH(sk, hash, saddr, daddr, ports, dif, ns))
    goto hit; /* You sunk my battleship! */
  }
  /* Must check for a TIME_WAIT'er before going to listener hash. */
```

```
@@ -83,6 +84,7 @@ struct sock *__inet6_lookup_established(
   const struct inet_timewait_sock *tw = inet_twsk(sk);

   if(*((__u32 *)&(tw->tw_dport)) == ports &&
+     net_ns_match(sk->sk_net_ns, ns)  &&
      sk->sk_family  == PF_INET6) {
    const struct inet6_timewait_sock *tw6 = inet6_twsk(sk);

@@ -107,12 +109,15 @@ struct sock *inet6_lookup_listener(struc
        const unsigned short hnum, const int dif)
 {
  struct sock *sk;
+ struct net_namespace *ns = current_net_ns;
  const struct hlist_node *node;
  struct sock *result = NULL;
  int score, hiscore = 0;

  read_lock(&hashinfo->lhash_lock);
- sk_for_each(sk, node, &hashinfo->listening_hash[inet_lhashfn(hnum)]) {
+ sk_for_each(sk, node, &hashinfo->listening_hash[inet_lhashfn(hnum, ns)]) {
+  if (!net_ns_match(sk->sk_net_ns, ns))
+   continue;
   if (inet_sk(sk)->num == hnum && sk->sk_family == PF_INET6) {
    const struct ipv6_pinfo *np = inet6_sk(sk);

@@ -172,8 +177,9 @@ static int __inet6_check_established(str
  const struct in6_addr *saddr = &np->daddr;
  const int dif = sk->sk_bound_dev_if;
  const u32 ports = INET_COMBINED_PORTS(inet->dport, lport);
+ struct net_namespace *ns = current_net_ns;
  const unsigned int hash = inet6_ehashfn(daddr, inet->num, saddr,
-      inet->dport);
+      inet->dport, ns);
  struct inet_ehash_bucket *head = inet_ehash_bucket(hinfo, hash);
  struct sock *sk2;
  const struct hlist_node *node;
@@ -190,6 +196,7 @@ static int __inet6_check_established(str

   if(*((__u32 *)&(tw->tw_dport)) == ports   &&
      sk2->sk_family        == PF_INET6  &&
+     net_ns_match(sk2->sk_net_ns, ns)   &&
      ipv6_addr_equal(&tw6->tw_v6_daddr, saddr)  &&
      ipv6_addr_equal(&tw6->tw_v6_rcv_saddr, daddr) &&
      sk2->sk_bound_dev_if == sk->sk_bound_dev_if) {
@@ -203,7 +210,7 @@ static int __inet6_check_established(str

  /* And established part... */
  sk_for_each(sk2, node, &head->chain) {
```

```
-  if (INET6_MATCH(sk2, hash, saddr, daddr, ports, dif))
+  if (INET6_MATCH(sk2, hash, saddr, daddr, ports, dif, ns))
     goto not_unique;
   }

@@ -249,6 +256,7 @@ int inet6_hash_connect(struct inet_timew
 {
   struct inet_hashinfo *hinfo = death_row->hashinfo;
   const unsigned short snum = inet_sk(sk)->num;
+ struct net_namespace *ns = current_net_ns;
    struct inet_bind_hashbucket *head;
    struct inet_bind_bucket *tb;
   int ret;
@@ -266,7 +274,8 @@ int inet6_hash_connect(struct inet_timew
     local_bh_disable();
    for (i = 1; i <= range; i++) {
     port = low + (i + offset) % range;
-    head = &hinfo->bhash[inet_bhashfn(port, hinfo->bhash_size)];
+    head = &hinfo->bhash[inet_bhashfn(port, ns,
+       hinfo->bhash_size)];
     spin_lock(&head->lock);

     /* Does not bother with rcv_saddr checks,
@@ -274,6 +283,8 @@ int inet6_hash_connect(struct inet_timew
      * unique enough.
      */
    inet_bind_bucket_for_each(tb, node, &head->chain) {
+    if (!net_ns_match(tb->net_ns, ns))
+     continue;
      if (tb->port == port) {
       BUG_TRAP(!hlist_empty(&tb->owners));
       if (tb->fastreuse >= 0)
@@ -322,7 +333,7 @@ ok:
   goto out;
   }

-  head = &hinfo->bhash[inet_bhashfn(snum, hinfo->bhash_size)];
+  head = &hinfo->bhash[inet_bhashfn(snum, ns, hinfo->bhash_size)];
   tb   = inet_csk(sk)->icsk_bind_hash;
  spin_lock_bh(&head->lock);

--- ./net/ipv6/raw.c.venssock Mon Aug 14 17:04:10 2006
+++ ./net/ipv6/raw.c Tue Aug 15 13:38:47 2006
@@ -87,11 +87,15 @@ struct sock *__raw_v6_lookup(struct sock
 {
  struct hlist_node *node;
  int is_multicast = ipv6_addr_is_multicast(loc_addr);
+ struct net_namespace *ns = current_net_ns;
```

```
  sk_for_each_from(sk, node)
   if (inet_sk(sk)->num == num) {
    struct ipv6_pinfo *np = inet6_sk(sk);

+   if (!net_ns_match(sk->sk_net_ns, ns))
+    continue;
+
    if (!ipv6_addr_any(&np->daddr) &&
        !ipv6_addr_equal(&np->daddr, rmt_addr))
     continue;
--- ./net/ipv6/udp.c.venssock Mon Aug 14 17:04:10 2006
+++ ./net/ipv6/udp.c Tue Aug 15 13:38:47 2006
@@ -68,6 +68,7 @@ static int udp_v6_get_port(struct sock *
 {
  struct sock *sk2;
  struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;

  write_lock_bh(&udp_hash_lock);
  if (snum == 0) {
@@ -82,7 +83,7 @@ static int udp_v6_get_port(struct sock *
    int size;
    struct hlist_head *list;

-   list = &udp_hash[result & (UDP_HTABLE_SIZE - 1)];
+   list = &udp_hash[udp_hashfn(result, ns)];
    if (hlist_empty(list)) {
     if (result > sysctl_local_port_range[1])
      result = sysctl_local_port_range[0] +
@@ -104,7 +105,7 @@ static int udp_v6_get_port(struct sock *
     result = sysctl_local_port_range[0]
      + ((result - sysctl_local_port_range[0]) &
         (UDP_HTABLE_SIZE - 1));
-   if (!udp_lport_inuse(result))
+   if (!udp_lport_inuse(result, ns))
     break;
   }
   if (i >= (1 << 16) / UDP_HTABLE_SIZE)
@@ -112,10 +113,10 @@ static int udp_v6_get_port(struct sock *
 gotit:
   udp_port_rover = snum = result;
  } else {
-  sk_for_each(sk2, node,
-       &udp_hash[snum & (UDP_HTABLE_SIZE - 1)]) {
+  sk_for_each(sk2, node, &udp_hash[udp_hashfn(snum, ns)]) {
    if (inet_sk(sk2)->num == snum &&
        sk2 != sk &&
```

```
+       net_ns_match(sk2->sk_net_ns, ns) &&
        (!sk2->sk_bound_dev_if ||
         !sk->sk_bound_dev_if ||
         sk2->sk_bound_dev_if == sk->sk_bound_dev_if) &&
@@ -127,7 +128,7 @@ gotit:

  inet_sk(sk)->num = snum;
  if (sk_unhashed(sk)) {
-  sk_add_node(sk, &udp_hash[snum & (UDP_HTABLE_SIZE - 1)]);
+  sk_add_node(sk, &udp_hash[udp_hashfn(snum, ns)]);
    sock_prot_inc_use(sk->sk_prot);
  }
  write_unlock_bh(&udp_hash_lock);
@@ -158,13 +159,16 @@ static struct sock *udp_v6_lookup(struct
 {
  struct sock *sk, *result = NULL;
  struct hlist_node *node;
+ struct net_namespace *ns = current_net_ns;
  unsigned short hnum = ntohs(dport);
  int badness = -1;

    read_lock(&udp_hash_lock);
- sk_for_each(sk, node, &udp_hash[hnum & (UDP_HTABLE_SIZE - 1)]) {
+ sk_for_each(sk, node, &udp_hash[udp_hashfn(hnum, ns)]) {
    struct inet_sock *inet = inet_sk(sk);

+   if (!net_ns_match(sk->sk_net_ns, ns))
+     continue;
    if (inet->num == hnum && sk->sk_family == PF_INET6) {
     struct ipv6_pinfo *np = inet6_sk(sk);
     int score = 0;
@@ -372,6 +376,7 @@ static struct sock *udp_v6_mcast_next(st
 {
  struct hlist_node *node;
  struct sock *s = sk;
+ struct net_namespace *ns = current_net_ns;
  unsigned short num = ntohs(loc_port);

  sk_for_each_from(s, node) {
@@ -383,6 +388,8 @@ static struct sock *udp_v6_mcast_next(st
      if (inet->dport != rmt_port)
       continue;
     }
+    if (!net_ns_match(sk->sk_net_ns, ns))
+     continue;
     if (!ipv6_addr_any(&np->daddr) &&
         !ipv6_addr_equal(&np->daddr, rmt_addr))
      continue;
```

```
@@ -414,7 +421,7 @@ static void udpv6_mcast_deliver(struct u
  int dif;

  read_lock(&udp_hash_lock);
- sk = sk_head(&udp_hash[ntohs(uh->dest) & (UDP_HTABLE_SIZE - 1)]);
+ sk = sk_head(&udp_hash[udp_hashfn(uh->dest, current_net_ns)]);
  dif = skb->dev->ifindex;
  sk = udp_v6_mcast_next(sk, uh->dest, daddr, uh->source, saddr, dif);
  if (!sk) {
```