Subject: [PATCH 0/8] Cleanup/fix the sk_alloc() call
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:38:07 GMT
View Forum Message <> Reply to Message

The sk_alloc() function suffers from two problems:
1 (major). The error path is not clean in it - if the security
   call fails, the net namespace is not put, if the try_module_get
   fails additionally the security context is not released;
2 (minor). The zero_it argument is misleading, as it doesn't just
   zeroes it, but performs some extra setup. Besides this argument
   is used only in one place - in the sk_clone().

So this set fixes these problems and performs some additional
cleanup.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Subject: [PATCH 1/8] Move the sock_copy() from the header
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:40:03 GMT
View Forum Message <> Reply to Message

The sock_copy() call is not used outside the sock.c file,
so just move it into a sock.c

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/include/net/sock.h b/include/net/sock.h
index 43fc3fa..ecad7b4 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -993,20 +993,6 @@ static inline void sock_graft(struct sock *sk, struct socket *parent)
  write_unlock_bh(&sk->sk_callback_lock);
 }

-static inline void sock_copy(struct sock *nsk, const struct sock *osk)
-{
-#ifdef CONFIG_SECURITY_NETWORK
- void *sptr = nsk->sk_security;
-#endif
-
- memcpy(nsk, osk, osk->sk_prot->obj_size);
- get_net(nsk->sk_net);
-#ifdef CONFIG_SECURITY_NETWORK
- nsk->sk_security = sptr;
- security_sk_clone(osk, nsk);
```

```
-#endif
-}
-
 extern int sock_i_uid(struct sock *sk);
 extern unsigned long sock_i_ino(struct sock *sk);

diff --git a/net/core/sock.c b/net/core/sock.c
index bba9949..fdacf9c 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -857,6 +857,20 @@ static inline void sock_lock_init(struct sock *sk)
    af_family_keys + sk->sk_family);
 }

+static void sock_copy(struct sock *nsk, const struct sock *osk)
+{
+#ifdef CONFIG_SECURITY_NETWORK
+ void *sptr = nsk->sk_security;
+#endif
+
+ memcpy(nsk, osk, osk->sk_prot->obj_size);
+ get_net(nsk->sk_net);
+#ifdef CONFIG_SECURITY_NETWORK
+ nsk->sk_security = sptr;
+ security_sk_clone(osk, nsk);
+#endif
+}
+
 /**
  * sk_alloc - All socket objects are allocated here
  * @net: the applicable net namespace
--
1.5.3.4
```

Subject: [PATCH 2/8] Move the get_net() from sock_copy()
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:42:40 GMT
View Forum Message <> Reply to Message

The sock_copy() is supposed to just clone the socket. In a perfect
world it has to be just memcpy, but we have to handle the security
mark correctly. All the extra setup must be performed in sk_clone()
call, so move the get_net() into more proper place.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/net/core/sock.c b/net/core/sock.c
index fdacf9c..9c2dbfa 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -864,7 +864,6 @@ static void sock_copy(struct sock *nsk, const struct sock *osk)
 #endif

  memcpy(nsk, osk, osk->sk_prot->obj_size);
- get_net(nsk->sk_net);
 #ifdef CONFIG_SECURITY_NETWORK
  nsk->sk_security = sptr;
  security_sk_clone(osk, nsk);
@@ -958,6 +957,7 @@ struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
   sock_copy(newsk, sk);

   /* SANITY */
+  get_net(newsk->sk_net);
   sk_node_init(&newsk->sk_node);
   sock_lock_init(newsk);
   bh_lock_sock(newsk);
--
1.5.3.4
```

Subject: [PATCH 3/8] Cleanup the allocation/freeing of the sock object
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:45:03 GMT
View Forum Message <> Reply to Message

The sock object is allocated either from the generic cache with
the kmalloc, or from the proc->slab cache.

Move this logic into an isolated set of helpers and make the
sk_alloc/sk_free look a bit nicer.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/net/core/sock.c b/net/core/sock.c
index 9c2dbfa..7c2e3db 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -870,6 +870,31 @@ static void sock_copy(struct sock *nsk, const struct sock *osk)
 #endif
 }

+static struct sock *sk_prot_alloc(struct proto *prot, gfp_t priority)
+{
```

```
+ struct sock *sk;
+ struct kmem_cache *slab;
+
+ slab = prot->slab;
+ if (slab != NULL)
+  sk = kmem_cache_alloc(slab, priority);
+ else
+  sk = kmalloc(prot->obj_size, priority);
+
+ return sk;
+}
+
+static void sk_prot_free(struct proto *prot, struct sock *sk)
+{
+ struct kmem_cache *slab;
+
+ slab = prot->slab;
+ if (slab != NULL)
+  kmem_cache_free(slab, sk);
+ else
+  kfree(sk);
+}
+
 /**
  * sk_alloc - All socket objects are allocated here
  * @net: the applicable net namespace
@@ -881,14 +906,9 @@ static void sock_copy(struct sock *nsk, const struct sock *osk)
 struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
         struct proto *prot, int zero_it)
 {
- struct sock *sk = NULL;
- struct kmem_cache *slab = prot->slab;
-
- if (slab != NULL)
-  sk = kmem_cache_alloc(slab, priority);
- else
-  sk = kmalloc(prot->obj_size, priority);
+ struct sock *sk;

+ sk = sk_prot_alloc(prot, priority);
  if (sk) {
   if (zero_it) {
    memset(sk, 0, prot->obj_size);
@@ -911,10 +931,7 @@ struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
  return sk;

 out_free:
- if (slab != NULL)
```

```
-  kmem_cache_free(slab, sk);
- else
-  kfree(sk);
+ sk_prot_free(prot, sk);
  return NULL;
 }

@@ -940,10 +957,7 @@ void sk_free(struct sock *sk)

  security_sk_free(sk);
  put_net(sk->sk_net);
- if (sk->sk_prot_creator->slab != NULL)
-  kmem_cache_free(sk->sk_prot_creator->slab, sk);
- else
-  kfree(sk);
+ sk_prot_free(sk->sk_prot_creator, sk);
  module_put(owner);
 }

--
1.5.3.4
```

Subject: [PATCH 4/8] Auto-zero the allocated sock object
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:46:56 GMT
View Forum Message <> Reply to Message

We have a __GFP_ZERO flag that allocates a zeroed chunk of memory.
Use it in the sk_alloc() and avoid a hand-made memset().

This is a temporary patch that will help us in the nearest future :)

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/net/core/sock.c b/net/core/sock.c
index 7c2e3db..21fc79b 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -908,10 +908,12 @@ struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
 {
  struct sock *sk;

+ if (zero_it)
+  priority |= __GFP_ZERO;
+
  sk = sk_prot_alloc(prot, priority);
```

```
  if (sk) {
   if (zero_it) {
-   memset(sk, 0, prot->obj_size);
    sk->sk_family = family;
    /*
     * See comment in struct sock definition to understand
--
1.5.3.4
```

Subject: [PATCH 5/8] Move some core sock setup into sk_prot_alloc
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:49:59 GMT
View Forum Message <> Reply to Message

The security_sk_alloc() and the module_get is a part of the
object allocations - move it in the proper place.

Note, that since we do not reset the newly allocated sock
in the sk_alloc() (memset() is removed with the previous
patch) we can safely do this.

Also fix the error path in sk_prot_alloc() - release the security
context if needed.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/net/core/sock.c b/net/core/sock.c
index 21fc79b..e7537e4 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -870,7 +870,8 @@ static void sock_copy(struct sock *nsk, const struct sock *osk)
 #endif
 }

-static struct sock *sk_prot_alloc(struct proto *prot, gfp_t priority)
+static struct sock *sk_prot_alloc(struct proto *prot, gfp_t priority,
+  int family)
 {
  struct sock *sk;
  struct kmem_cache *slab;
@@ -881,18 +882,40 @@ static struct sock *sk_prot_alloc(struct proto *prot, gfp_t priority)
  else
   sk = kmalloc(prot->obj_size, priority);

+ if (sk != NULL) {
+  if (security_sk_alloc(sk, family, priority))
```

```
+   goto out_free;
+
+  if (!try_module_get(prot->owner))
+   goto out_free_sec;
+ }
+
  return sk;
+
+out_free_sec:
+ security_sk_free(sk);
+out_free:
+ if (slab != NULL)
+  kmem_cache_free(slab, sk);
+ else
+  kfree(sk);
+ return NULL;
 }

 static void sk_prot_free(struct proto *prot, struct sock *sk)
 {
  struct kmem_cache *slab;
-
+ struct module *owner;
+
+ owner = prot->owner;
  slab = prot->slab;
+
+ security_sk_free(sk);
  if (slab != NULL)
   kmem_cache_free(slab, sk);
  else
   kfree(sk);
+ module_put(owner);
 }

 /**
@@ -911,7 +934,7 @@ struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
  if (zero_it)
   priority |= __GFP_ZERO;

- sk = sk_prot_alloc(prot, priority);
+ sk = sk_prot_alloc(prot, priority, family);
  if (sk) {
   if (zero_it) {
    sk->sk_family = family;
@@ -923,24 +946,14 @@ struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
    sock_lock_init(sk);
    sk->sk_net = get_net(net);
```

```
   }
-
-  if (security_sk_alloc(sk, family, priority))
-   goto out_free;
-
-  if (!try_module_get(prot->owner))
-   goto out_free;
  }
- return sk;

-out_free:
- sk_prot_free(prot, sk);
- return NULL;
+ return sk;
 }

 void sk_free(struct sock *sk)
 {
  struct sk_filter *filter;
- struct module *owner = sk->sk_prot_creator->owner;

  if (sk->sk_destruct)
   sk->sk_destruct(sk);
@@ -957,10 +970,8 @@ void sk_free(struct sock *sk)
   printk(KERN_DEBUG "%s: optmem leakage (%d bytes) detected.\n",
          __FUNCTION__, atomic_read(&sk->sk_omem_alloc));

- security_sk_free(sk);
  put_net(sk->sk_net);
  sk_prot_free(sk->sk_prot_creator, sk);
- module_put(owner);
 }

 struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
--
1.5.3.4
```

Subject: [PATCH 6/8] Make the sk_clone() lighter
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:52:32 GMT
View Forum Message <> Reply to Message

The sk_prot_alloc() already performs all the stuff needed by the
sk_clone(). Besides, the sk_prot_alloc() requires almost twice
less arguments than the sk_alloc() does, so call the sk_prot_alloc()
saving the stack a bit.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/net/core/sock.c b/net/core/sock.c
index e7537e4..c032f48 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -976,8 +976,9 @@ void sk_free(struct sock *sk)

 struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
 {
- struct sock *newsk = sk_alloc(sk->sk_net, sk->sk_family, priority, sk->sk_prot, 0);
-
+ struct sock *newsk;
+
+ newsk = sk_prot_alloc(sk->sk_prot, priority, sk->sk_family);
  if (newsk != NULL) {
   struct sk_filter *filter;

--
1.5.3.4
```

Subject: [PATCH 7/8] Remove bogus zero_it argument from sk_alloc
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:55:19 GMT
View Forum Message <> Reply to Message

At this point nobody calls the sk_alloc(() with zero_it == 0,
so remove unneeded checks from it.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/net/core/sock.c b/net/core/sock.c
index c032f48..77575c3 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -931,21 +931,16 @@ struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
 {
  struct sock *sk;

- if (zero_it)
-  priority |= __GFP_ZERO;
-
- sk = sk_prot_alloc(prot, priority, family);
+ sk = sk_prot_alloc(prot, priority | __GFP_ZERO, family);
  if (sk) {
```

```
-  if (zero_it) {
-   sk->sk_family = family;
-   /*
-    * See comment in struct sock definition to understand
-    * why we need sk_prot_creator -acme
-    */
-   sk->sk_prot = sk->sk_prot_creator = prot;
-   sock_lock_init(sk);
-   sk->sk_net = get_net(net);
-  }
+  sk->sk_family = family;
+  /*
+   * See comment in struct sock definition to understand
+   * why we need sk_prot_creator -acme
+   */
+  sk->sk_prot = sk->sk_prot_creator = prot;
+  sock_lock_init(sk);
+  sk->sk_net = get_net(net);
   }

  return sk;
--
1.5.3.4
```

Subject: [PATCH 8/8] Forget the zero_it argument of sk_alloc()
Posted by Pavel Emelianov on Wed, 31 Oct 2007 12:57:18 GMT
View Forum Message <> Reply to Message

Finally, the zero_it argument can be completely removed from
the callers and from the function prototype.

Besides, fix the checkpatch.pl warnings about using the
assignments inside if-s.

This patch is rather big, and it is a part of the previous one.
I splitted it wishing to make the patches more readable. Hope
this particular split helped.

Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

```
diff --git a/drivers/net/pppoe.c b/drivers/net/pppoe.c
index 8936ed3..a005d8f 100644
--- a/drivers/net/pppoe.c
+++ b/drivers/net/pppoe.c
@@ -491,7 +491,7 @@ static int pppoe_create(struct net *net, struct socket *sock)
```

```
  int error = -ENOMEM;
  struct sock *sk;

- sk = sk_alloc(net, PF_PPPOX, GFP_KERNEL, &pppoe_sk_proto, 1);
+ sk = sk_alloc(net, PF_PPPOX, GFP_KERNEL, &pppoe_sk_proto);
  if (!sk)
   goto out;

diff --git a/drivers/net/pppol2tp.c b/drivers/net/pppol2tp.c
index 921d4ef..f8904fd 100644
--- a/drivers/net/pppol2tp.c
+++ b/drivers/net/pppol2tp.c
@@ -1416,7 +1416,7 @@ static int pppol2tp_create(struct net *net, struct socket *sock)
  int error = -ENOMEM;
  struct sock *sk;

- sk = sk_alloc(net, PF_PPPOX, GFP_KERNEL, &pppol2tp_sk_proto, 1);
+ sk = sk_alloc(net, PF_PPPOX, GFP_KERNEL, &pppol2tp_sk_proto);
  if (!sk)
   goto out;

diff --git a/include/net/sock.h b/include/net/sock.h
index ecad7b4..20de3fa 100644
--- a/include/net/sock.h
+++ b/include/net/sock.h
@@ -779,7 +779,7 @@ extern void FASTCALL(release_sock(struct sock *sk));

 extern struct sock  *sk_alloc(struct net *net, int family,
        gfp_t priority,
-       struct proto *prot, int zero_it);
+       struct proto *prot);
 extern void   sk_free(struct sock *sk);
 extern struct sock  *sk_clone(const struct sock *sk,
        const gfp_t priority);
diff --git a/net/appletalk/ddp.c b/net/appletalk/ddp.c
index 7c0b515..e0d37d6 100644
--- a/net/appletalk/ddp.c
+++ b/net/appletalk/ddp.c
@@ -1044,7 +1044,7 @@ static int atalk_create(struct net *net, struct socket *sock, int protocol)
  if (sock->type != SOCK_RAW && sock->type != SOCK_DGRAM)
   goto out;
  rc = -ENOMEM;
- sk = sk_alloc(net, PF_APPLETALK, GFP_KERNEL, &ddp_proto, 1);
+ sk = sk_alloc(net, PF_APPLETALK, GFP_KERNEL, &ddp_proto);
  if (!sk)
   goto out;
  rc = 0;
diff --git a/net/atm/common.c b/net/atm/common.c
```

```
index e166d9e..eba09a0 100644
--- a/net/atm/common.c
+++ b/net/atm/common.c
@@ -133,7 +133,7 @@ int vcc_create(struct net *net, struct socket *sock, int protocol, int family)
  sock->sk = NULL;
  if (sock->type == SOCK_STREAM)
   return -EINVAL;
- sk = sk_alloc(net, family, GFP_KERNEL, &vcc_proto, 1);
+ sk = sk_alloc(net, family, GFP_KERNEL, &vcc_proto);
  if (!sk)
   return -ENOMEM;
  sock_init_data(sock, sk);
diff --git a/net/ax25/af_ax25.c b/net/ax25/af_ax25.c
index 993e5c7..8378afd 100644
--- a/net/ax25/af_ax25.c
+++ b/net/ax25/af_ax25.c
@@ -836,7 +836,8 @@ static int ax25_create(struct net *net, struct socket *sock, int protocol)
   return -ESOCKTNOSUPPORT;
  }

- if ((sk = sk_alloc(net, PF_AX25, GFP_ATOMIC, &ax25_proto, 1)) == NULL)
+ sk = sk_alloc(net, PF_AX25, GFP_ATOMIC, &ax25_proto);
+ if (sk == NULL)
   return -ENOMEM;

  ax25 = sk->sk_protinfo = ax25_create_cb();
@@ -861,7 +862,8 @@ struct sock *ax25_make_new(struct sock *osk, struct ax25_dev
*ax25_dev)
  struct sock *sk;
  ax25_cb *ax25, *oax25;

- if ((sk = sk_alloc(osk->sk_net, PF_AX25, GFP_ATOMIC, osk->sk_prot, 1)) == NULL)
+ sk = sk_alloc(osk->sk_net, PF_AX25, GFP_ATOMIC, osk->sk_prot);
+ if (sk == NULL)
   return NULL;

  if ((ax25 = ax25_create_cb()) == NULL) {
diff --git a/net/bluetooth/bnep/sock.c b/net/bluetooth/bnep/sock.c
index f718965..9ebd3c6 100644
--- a/net/bluetooth/bnep/sock.c
+++ b/net/bluetooth/bnep/sock.c
@@ -213,7 +213,7 @@ static int bnep_sock_create(struct net *net, struct socket *sock, int
protocol)
  if (sock->type != SOCK_RAW)
   return -ESOCKTNOSUPPORT;

- sk = sk_alloc(net, PF_BLUETOOTH, GFP_ATOMIC, &bnep_proto, 1);
+ sk = sk_alloc(net, PF_BLUETOOTH, GFP_ATOMIC, &bnep_proto);
```

```
  if (!sk)
   return -ENOMEM;

diff --git a/net/bluetooth/cmtp/sock.c b/net/bluetooth/cmtp/sock.c
index cf700c2..783edab 100644
--- a/net/bluetooth/cmtp/sock.c
+++ b/net/bluetooth/cmtp/sock.c
@@ -204,7 +204,7 @@ static int cmtp_sock_create(struct net *net, struct socket *sock, int
protocol)
  if (sock->type != SOCK_RAW)
   return -ESOCKTNOSUPPORT;

- sk = sk_alloc(net, PF_BLUETOOTH, GFP_ATOMIC, &cmtp_proto, 1);
+ sk = sk_alloc(net, PF_BLUETOOTH, GFP_ATOMIC, &cmtp_proto);
  if (!sk)
   return -ENOMEM;

diff --git a/net/bluetooth/hci_sock.c b/net/bluetooth/hci_sock.c
index 8825102..1499132 100644
--- a/net/bluetooth/hci_sock.c
+++ b/net/bluetooth/hci_sock.c
@@ -645,7 +645,7 @@ static int hci_sock_create(struct net *net, struct socket *sock, int protocol)

  sock->ops = &hci_sock_ops;

- sk = sk_alloc(net, PF_BLUETOOTH, GFP_ATOMIC, &hci_sk_proto, 1);
+ sk = sk_alloc(net, PF_BLUETOOTH, GFP_ATOMIC, &hci_sk_proto);
  if (!sk)
   return -ENOMEM;

diff --git a/net/bluetooth/hidp/sock.c b/net/bluetooth/hidp/sock.c
index 1de2b6f..3292b95 100644
--- a/net/bluetooth/hidp/sock.c
+++ b/net/bluetooth/hidp/sock.c
@@ -255,7 +255,7 @@ static int hidp_sock_create(struct net *net, struct socket *sock, int
protocol)
  if (sock->type != SOCK_RAW)
   return -ESOCKTNOSUPPORT;

- sk = sk_alloc(net, PF_BLUETOOTH, GFP_ATOMIC, &hidp_proto, 1);
+ sk = sk_alloc(net, PF_BLUETOOTH, GFP_ATOMIC, &hidp_proto);
  if (!sk)
   return -ENOMEM;

diff --git a/net/bluetooth/l2cap.c b/net/bluetooth/l2cap.c
index 6fbbae7..477e052 100644
--- a/net/bluetooth/l2cap.c
+++ b/net/bluetooth/l2cap.c
```

```
@@ -607,7 +607,7 @@ static struct sock *l2cap_sock_alloc(struct net *net, struct socket *sock,
int p
 {
  struct sock *sk;

- sk = sk_alloc(net, PF_BLUETOOTH, prio, &l2cap_proto, 1);
+ sk = sk_alloc(net, PF_BLUETOOTH, prio, &l2cap_proto);
  if (!sk)
   return NULL;

diff --git a/net/bluetooth/rfcomm/sock.c b/net/bluetooth/rfcomm/sock.c
index 266b697..c46d510 100644
--- a/net/bluetooth/rfcomm/sock.c
+++ b/net/bluetooth/rfcomm/sock.c
@@ -287,7 +287,7 @@ static struct sock *rfcomm_sock_alloc(struct net *net, struct socket *sock,
int
  struct rfcomm_dlc *d;
  struct sock *sk;

- sk = sk_alloc(net, PF_BLUETOOTH, prio, &rfcomm_proto, 1);
+ sk = sk_alloc(net, PF_BLUETOOTH, prio, &rfcomm_proto);
  if (!sk)
   return NULL;

diff --git a/net/bluetooth/sco.c b/net/bluetooth/sco.c
index 82d0dfd..93ad1aa 100644
--- a/net/bluetooth/sco.c
+++ b/net/bluetooth/sco.c
@@ -421,7 +421,7 @@ static struct sock *sco_sock_alloc(struct net *net, struct socket *sock, int
pro
 {
  struct sock *sk;

- sk = sk_alloc(net, PF_BLUETOOTH, prio, &sco_proto, 1);
+ sk = sk_alloc(net, PF_BLUETOOTH, prio, &sco_proto);
  if (!sk)
   return NULL;

diff --git a/net/core/sock.c b/net/core/sock.c
index 6046fc6..12ad206 100644
--- a/net/core/sock.c
+++ b/net/core/sock.c
@@ -927,7 +927,7 @@ static void sk_prot_free(struct proto *prot, struct sock *sk)
  * @zero_it: if we should zero the newly allocated sock
  */
 struct sock *sk_alloc(struct net *net, int family, gfp_t priority,
-       struct proto *prot, int zero_it)
+       struct proto *prot)
```

```
 {
  struct sock *sk;

diff --git a/net/decnet/af_decnet.c b/net/decnet/af_decnet.c
index aabe98d..57d5749 100644
--- a/net/decnet/af_decnet.c
+++ b/net/decnet/af_decnet.c
@@ -474,7 +474,7 @@ static struct proto dn_proto = {
 static struct sock *dn_alloc_sock(struct net *net, struct socket *sock, gfp_t gfp)
 {
  struct dn_scp *scp;
- struct sock *sk = sk_alloc(net, PF_DECnet, gfp, &dn_proto, 1);
+ struct sock *sk = sk_alloc(net, PF_DECnet, gfp, &dn_proto);

  if  (!sk)
   goto out;
diff --git a/net/econet/af_econet.c b/net/econet/af_econet.c
index 9cae16b..f70df07 100644
--- a/net/econet/af_econet.c
+++ b/net/econet/af_econet.c
@@ -624,7 +624,7 @@ static int econet_create(struct net *net, struct socket *sock, int protocol)
  sock->state = SS_UNCONNECTED;

  err = -ENOBUFS;
- sk = sk_alloc(net, PF_ECONET, GFP_KERNEL, &econet_proto, 1);
+ sk = sk_alloc(net, PF_ECONET, GFP_KERNEL, &econet_proto);
  if (sk == NULL)
   goto out;

diff --git a/net/ipv4/af_inet.c b/net/ipv4/af_inet.c
index 621b128..d2f22e7 100644
--- a/net/ipv4/af_inet.c
+++ b/net/ipv4/af_inet.c
@@ -323,7 +323,7 @@ lookup_protocol:
  BUG_TRAP(answer_prot->slab != NULL);

  err = -ENOBUFS;
- sk = sk_alloc(net, PF_INET, GFP_KERNEL, answer_prot, 1);
+ sk = sk_alloc(net, PF_INET, GFP_KERNEL, answer_prot);
  if (sk == NULL)
   goto out;

diff --git a/net/ipv6/af_inet6.c b/net/ipv6/af_inet6.c
index 1b1caf3..ecbd388 100644
--- a/net/ipv6/af_inet6.c
+++ b/net/ipv6/af_inet6.c
@@ -162,7 +162,7 @@ lookup_protocol:
  BUG_TRAP(answer_prot->slab != NULL);
```

```
  err = -ENOBUFS;
- sk = sk_alloc(net, PF_INET6, GFP_KERNEL, answer_prot, 1);
+ sk = sk_alloc(net, PF_INET6, GFP_KERNEL, answer_prot);
  if (sk == NULL)
   goto out;

diff --git a/net/ipx/af_ipx.c b/net/ipx/af_ipx.c
index 29b063d..a195a66 100644
--- a/net/ipx/af_ipx.c
+++ b/net/ipx/af_ipx.c
@@ -1381,7 +1381,7 @@ static int ipx_create(struct net *net, struct socket *sock, int protocol)
   goto out;

  rc = -ENOMEM;
- sk = sk_alloc(net, PF_IPX, GFP_KERNEL, &ipx_proto, 1);
+ sk = sk_alloc(net, PF_IPX, GFP_KERNEL, &ipx_proto);
  if (!sk)
   goto out;
 #ifdef IPX_REFCNT_DEBUG
diff --git a/net/irda/af_irda.c b/net/irda/af_irda.c
index 0328ae2..48ce59a 100644
--- a/net/irda/af_irda.c
+++ b/net/irda/af_irda.c
@@ -1078,7 +1078,7 @@ static int irda_create(struct net *net, struct socket *sock, int protocol)
  }

  /* Allocate networking socket */
- sk = sk_alloc(net, PF_IRDA, GFP_ATOMIC, &irda_proto, 1);
+ sk = sk_alloc(net, PF_IRDA, GFP_ATOMIC, &irda_proto);
  if (sk == NULL)
   return -ENOMEM;

diff --git a/net/iucv/af_iucv.c b/net/iucv/af_iucv.c
index 43e01c8..aef6645 100644
--- a/net/iucv/af_iucv.c
+++ b/net/iucv/af_iucv.c
@@ -216,7 +216,7 @@ static struct sock *iucv_sock_alloc(struct socket *sock, int proto, gfp_t
prio)
 {
  struct sock *sk;

- sk = sk_alloc(&init_net, PF_IUCV, prio, &iucv_proto, 1);
+ sk = sk_alloc(&init_net, PF_IUCV, prio, &iucv_proto);
  if (!sk)
   return NULL;

diff --git a/net/key/af_key.c b/net/key/af_key.c
```

```
index 266f112..10c89d4 100644
--- a/net/key/af_key.c
+++ b/net/key/af_key.c
@@ -152,7 +152,7 @@ static int pfkey_create(struct net *net, struct socket *sock, int protocol)
   return -EPROTONOSUPPORT;

  err = -ENOMEM;
- sk = sk_alloc(net, PF_KEY, GFP_KERNEL, &key_proto, 1);
+ sk = sk_alloc(net, PF_KEY, GFP_KERNEL, &key_proto);
  if (sk == NULL)
   goto out;

diff --git a/net/llc/llc_conn.c b/net/llc/llc_conn.c
index 8ebc276..5c0b484 100644
--- a/net/llc/llc_conn.c
+++ b/net/llc/llc_conn.c
@@ -869,7 +869,7 @@ static void llc_sk_init(struct sock* sk)
  */
 struct sock *llc_sk_alloc(struct net *net, int family, gfp_t priority, struct proto *prot)
 {
- struct sock *sk = sk_alloc(net, family, priority, prot, 1);
+ struct sock *sk = sk_alloc(net, family, priority, prot);

  if (!sk)
   goto out;
diff --git a/net/netlink/af_netlink.c b/net/netlink/af_netlink.c
index 4f994c0..2601712 100644
--- a/net/netlink/af_netlink.c
+++ b/net/netlink/af_netlink.c
@@ -396,7 +396,7 @@ static int __netlink_create(struct net *net, struct socket *sock,

  sock->ops = &netlink_ops;

- sk = sk_alloc(net, PF_NETLINK, GFP_KERNEL, &netlink_proto, 1);
+ sk = sk_alloc(net, PF_NETLINK, GFP_KERNEL, &netlink_proto);
  if (!sk)
   return -ENOMEM;

diff --git a/net/netrom/af_netrom.c b/net/netrom/af_netrom.c
index 3a4d479..972250c 100644
--- a/net/netrom/af_netrom.c
+++ b/net/netrom/af_netrom.c
@@ -423,7 +423,8 @@ static int nr_create(struct net *net, struct socket *sock, int protocol)
  if (sock->type != SOCK_SEQPACKET || protocol != 0)
   return -ESOCKTNOSUPPORT;

- if ((sk = sk_alloc(net, PF_NETROM, GFP_ATOMIC, &nr_proto, 1)) == NULL)
+ sk = sk_alloc(net, PF_NETROM, GFP_ATOMIC, &nr_proto);
```

```
+ if (sk  == NULL)
   return -ENOMEM;

  nr = nr_sk(sk);
@@ -465,7 +466,8 @@ static struct sock *nr_make_new(struct sock *osk)
  if (osk->sk_type != SOCK_SEQPACKET)
   return NULL;

- if ((sk = sk_alloc(osk->sk_net, PF_NETROM, GFP_ATOMIC, osk->sk_prot, 1)) == NULL)
+ sk = sk_alloc(osk->sk_net, PF_NETROM, GFP_ATOMIC, osk->sk_prot);
+ if (sk == NULL)
   return NULL;

  nr = nr_sk(sk);
diff --git a/net/packet/af_packet.c b/net/packet/af_packet.c
index d093650..4cb2dfb 100644
--- a/net/packet/af_packet.c
+++ b/net/packet/af_packet.c
@@ -995,7 +995,7 @@ static int packet_create(struct net *net, struct socket *sock, int protocol)
  sock->state = SS_UNCONNECTED;

  err = -ENOBUFS;
- sk = sk_alloc(net, PF_PACKET, GFP_KERNEL, &packet_proto, 1);
+ sk = sk_alloc(net, PF_PACKET, GFP_KERNEL, &packet_proto);
  if (sk == NULL)
   goto out;

diff --git a/net/rose/af_rose.c b/net/rose/af_rose.c
index 509defe..ed2d65c 100644
--- a/net/rose/af_rose.c
+++ b/net/rose/af_rose.c
@@ -513,7 +513,8 @@ static int rose_create(struct net *net, struct socket *sock, int protocol)
  if (sock->type != SOCK_SEQPACKET || protocol != 0)
   return -ESOCKTNOSUPPORT;

- if ((sk = sk_alloc(net, PF_ROSE, GFP_ATOMIC, &rose_proto, 1)) == NULL)
+ sk = sk_alloc(net, PF_ROSE, GFP_ATOMIC, &rose_proto);
+ if (sk == NULL)
   return -ENOMEM;

  rose = rose_sk(sk);
@@ -551,7 +552,8 @@ static struct sock *rose_make_new(struct sock *osk)
  if (osk->sk_type != SOCK_SEQPACKET)
   return NULL;

- if ((sk = sk_alloc(osk->sk_net, PF_ROSE, GFP_ATOMIC, &rose_proto, 1)) == NULL)
+ sk = sk_alloc(osk->sk_net, PF_ROSE, GFP_ATOMIC, &rose_proto);
+ if (sk == NULL)
```

```
   return NULL;

  rose = rose_sk(sk);
diff --git a/net/rxrpc/af_rxrpc.c b/net/rxrpc/af_rxrpc.c
index c680017..d638945 100644
--- a/net/rxrpc/af_rxrpc.c
+++ b/net/rxrpc/af_rxrpc.c
@@ -627,7 +627,7 @@ static int rxrpc_create(struct net *net, struct socket *sock, int protocol)
  sock->ops = &rxrpc_rpc_ops;
  sock->state = SS_UNCONNECTED;

- sk = sk_alloc(net, PF_RXRPC, GFP_KERNEL, &rxrpc_proto, 1);
+ sk = sk_alloc(net, PF_RXRPC, GFP_KERNEL, &rxrpc_proto);
  if (!sk)
   return -ENOMEM;

diff --git a/net/sctp/ipv6.c b/net/sctp/ipv6.c
index eb4deaf..7f31ff6 100644
--- a/net/sctp/ipv6.c
+++ b/net/sctp/ipv6.c
@@ -631,7 +631,7 @@ static struct sock *sctp_v6_create_accept_sk(struct sock *sk,
  struct ipv6_pinfo *newnp, *np = inet6_sk(sk);
  struct sctp6_sock *newsctp6sk;

- newsk = sk_alloc(sk->sk_net, PF_INET6, GFP_KERNEL, sk->sk_prot, 1);
+ newsk = sk_alloc(sk->sk_net, PF_INET6, GFP_KERNEL, sk->sk_prot);
  if (!newsk)
   goto out;

diff --git a/net/sctp/protocol.c b/net/sctp/protocol.c
index f5cd96f..40c1a47 100644
--- a/net/sctp/protocol.c
+++ b/net/sctp/protocol.c
@@ -552,7 +552,8 @@ static struct sock *sctp_v4_create_accept_sk(struct sock *sk,
 {
  struct inet_sock *inet = inet_sk(sk);
  struct inet_sock *newinet;
- struct sock *newsk = sk_alloc(sk->sk_net, PF_INET, GFP_KERNEL, sk->sk_prot, 1);
+ struct sock *newsk = sk_alloc(sk->sk_net, PF_INET, GFP_KERNEL,
+   sk->sk_prot);

  if (!newsk)
   goto out;
diff --git a/net/tipc/socket.c b/net/tipc/socket.c
index e36b4b5..6b79226 100644
--- a/net/tipc/socket.c
+++ b/net/tipc/socket.c
@@ -201,7 +201,7 @@ static int tipc_create(struct net *net, struct socket *sock, int protocol)
```

```
   return -EPROTOTYPE;
  }

- sk = sk_alloc(net, AF_TIPC, GFP_KERNEL, &tipc_proto, 1);
+ sk = sk_alloc(net, AF_TIPC, GFP_KERNEL, &tipc_proto);
  if (!sk) {
   tipc_deleteport(ref);
   return -ENOMEM;
diff --git a/net/unix/af_unix.c b/net/unix/af_unix.c
index 9163ec5..515e7a6 100644
--- a/net/unix/af_unix.c
+++ b/net/unix/af_unix.c
@@ -602,7 +602,7 @@ static struct sock * unix_create1(struct net *net, struct socket *sock)
  if (atomic_read(&unix_nr_socks) >= 2*get_max_files())
   goto out;

- sk = sk_alloc(net, PF_UNIX, GFP_KERNEL, &unix_proto, 1);
+ sk = sk_alloc(net, PF_UNIX, GFP_KERNEL, &unix_proto);
  if (!sk)
   goto out;

diff --git a/net/x25/af_x25.c b/net/x25/af_x25.c
index fc416f9..92cfe8e 100644
--- a/net/x25/af_x25.c
+++ b/net/x25/af_x25.c
@@ -472,7 +472,7 @@ static struct proto x25_proto = {
 static struct sock *x25_alloc_socket(struct net *net)
 {
  struct x25_sock *x25;
- struct sock *sk = sk_alloc(net, AF_X25, GFP_ATOMIC, &x25_proto, 1);
+ struct sock *sk = sk_alloc(net, AF_X25, GFP_ATOMIC, &x25_proto);

  if (!sk)
   goto out;
```

Subject: Re: [PATCH 0/8] Cleanup/fix the sk_alloc() call
Posted by Arnaldo Carvalho de M on Wed, 31 Oct 2007 13:15:05 GMT
View Forum Message <> Reply to Message

Em Wed, Oct 31, 2007 at 04:40:01PM +0300, Pavel Emelyanov escreveu:
> The sk_alloc() function suffers from two problems:
> 1 (major). The error path is not clean in it - if the security
>   call fails, the net namespace is not put, if the try_module_get
>   fails additionally the security context is not released;
> 2 (minor). The zero_it argument is misleading, as it doesn't just
>   zeroes it, but performs some extra setup. Besides this argument
>   is used only in one place - in the sk_clone().

>
> So this set fixes these problems and performs some additional
> cleanup.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

for the series:

Acked-by: Arnaldo Carvalho de Melo <acme@redhat.com>

Haven't tested, but it looks straightforward and conceptually sound,
thanks for improving the sk_prot infrastructure! :-)

Now we have just to make all the other protocols fill in the missing
sk->sk_prot-> methods (converting what is there now in socket->ops) so
that we can kill socket->ops and eliminate one level of indirection :-P

- Arnaldo

Subject: Re: [PATCH 0/8] Cleanup/fix the sk_alloc() call
Posted by Pavel Emelianov on Wed, 31 Oct 2007 13:30:24 GMT
View Forum Message <> Reply to Message

Arnaldo Carvalho de Melo wrote:
> Em Wed, Oct 31, 2007 at 04:40:01PM +0300, Pavel Emelyanov escreveu:
>> The sk_alloc() function suffers from two problems:
>> 1 (major). The error path is not clean in it - if the security
>>   call fails, the net namespace is not put, if the try_module_get
>>   fails additionally the security context is not released;
>> 2 (minor). The zero_it argument is misleading, as it doesn't just
>>   zeroes it, but performs some extra setup. Besides this argument
>>   is used only in one place - in the sk_clone().
>>
>> So this set fixes these problems and performs some additional
>> cleanup.
>>
>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> for the series:
>
> Acked-by: Arnaldo Carvalho de Melo <acme@redhat.com>

Thanks a lot :)

> Haven't tested, but it looks straightforward and conceptually sound,
> thanks for improving the sk_prot infrastructure! :-)

> Now we have just to make all the other protocols fill in the missing
> sk->sk_prot-> methods (converting what is there now in socket->ops) so
> that we can kill socket->ops and eliminate one level of indirection :-P

Do I get your idea right, that having the 'struct sock->ops' field is not
that good and the long-term TODO is to remove it (or smth similar)? Can you,
please, pour some more light on this, because I'm not yet very common with
the networking code, but I'm trying to learn it better by fixing obvious
bugs and cleaning the code.

> - Arnaldo

Thanks,
Pavel

Subject: Re: [PATCH 0/8] Cleanup/fix the sk_alloc() call
Posted by Arnaldo Carvalho de M on Wed, 31 Oct 2007 14:14:22 GMT
View Forum Message <> Reply to Message

Em Wed, Oct 31, 2007 at 05:32:20PM +0300, Pavel Emelyanov escreveu:
> Arnaldo Carvalho de Melo wrote:
> > Em Wed, Oct 31, 2007 at 04:40:01PM +0300, Pavel Emelyanov escreveu:
> >> The sk_alloc() function suffers from two problems:
> >> 1 (major). The error path is not clean in it - if the security
> >>   call fails, the net namespace is not put, if the try_module_get
> >>   fails additionally the security context is not released;
> >> 2 (minor). The zero_it argument is misleading, as it doesn't just
> >>   zeroes it, but performs some extra setup. Besides this argument
> >>   is used only in one place - in the sk_clone().
> >>
> >> So this set fixes these problems and performs some additional
> >> cleanup.
> >>
> >> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
> >
> > for the series:
> >
> > Acked-by: Arnaldo Carvalho de Melo <acme@redhat.com>
>
> Thanks a lot :)
>
> > Haven't tested, but it looks straightforward and conceptually sound,
> > thanks for improving the sk_prot infrastructure! :-)
>
> > Now we have just to make all the other protocols fill in the missing
> > sk->sk_prot-> methods (converting what is there now in socket->ops) so
> > that we can kill socket->ops and eliminate one level of indirection :-P

>
> Do I get your idea right, that having the 'struct sock->ops' field is not
> that good and the long-term TODO is to remove it (or smth similar)? Can you,
> please, pour some more light on this, because I'm not yet very common with
> the networking code, but I'm trying to learn it better by fixing obvious
> bugs and cleaning the code.

Start here:

```
const struct proto_ops inet_stream_ops = {
        .family            = PF_INET,
        .owner             = THIS_MODULE,
        .release           = inet_release,
        .bind              = inet_bind,
        .connect           = inet_stream_connect,
        .socketpair        = sock_no_socketpair,
        .accept            = inet_accept,
        .getname           = inet_getname,
        .poll              = tcp_poll,
        .ioctl             = inet_ioctl,
        .listen            = inet_listen,
        .shutdown          = inet_shutdown,
        .setsockopt        = sock_common_setsockopt,
        .getsockopt        = sock_common_getsockopt,
        .sendmsg           = tcp_sendmsg,
        .recvmsg           = sock_common_recvmsg,
        .mmap              = sock_no_mmap,
        .sendpage          = tcp_sendpage,
#ifdef CONFIG_COMPAT
        .compat_setsockopt = compat_sock_common_setsockopt,
        .compat_getsockopt = compat_sock_common_getsockopt,
#endif
};
```

Now look at all the "*_common_*" stuff, for instance:

```
int sock_common_recvmsg(struct kiocb *iocb, struct socket *sock,
                        struct msghdr *msg, size_t size, int flags)
{
        struct sock *sk = sock->sk;
        int addr_len = 0;
        int err;

        err = sk->sk_prot->recvmsg(iocb, sk, msg, size, flags & MSG_DONTWAIT,
                                   flags & ~MSG_DONTWAIT, &addr_len);
        if (err >= 0)
                msg->msg_namelen = addr_len;
        return err;
```

}

So if we made all protocols implement sk->sk_prot_recvmsg... got it?

And then look at the inet_* routines above, at least for LLC I was using
several unmodified.

Over the years the quality work is done on the mainstream protocols,
with the legacy ones lagging behind, so the more we share...

Anyway, look at my paper about it:

http://www.linuxsymposium.org/proceedings/reprints/Reprint-Melo-OLS2004.pdf

The DCCP paper also talks about this:

http://www.linuxinsight.com/files/ols2005/melo-reprint.pdf

- Arnaldo

Subject: Re: [PATCH 6/8] Make the sk_clone() lighter
Posted by davem on Thu, 01 Nov 2007 07:26:06 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 31 Oct 2007 16:54:34 +0300

> The sk_prot_alloc() already performs all the stuff needed by the
> sk_clone(). Besides, the sk_prot_alloc() requires almost twice
> less arguments than the sk_alloc() does, so call the sk_prot_alloc()
> saving the stack a bit.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>
> ---
>
> diff --git a/net/core/sock.c b/net/core/sock.c
> index e7537e4..c032f48 100644
> --- a/net/core/sock.c
> +++ b/net/core/sock.c
> @@ -976,8 +976,9 @@ void sk_free(struct sock *sk)
>
> struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
> {
> - struct sock *newsk = sk_alloc(sk->sk_net, sk->sk_family, priority, sk->sk_prot, 0);
> -
> + struct sock *newsk;

> +
> + newsk = sk_prot_alloc(sk->sk_prot, priority, sk->sk_family);
>   if (newsk != NULL) {
>   struct sk_filter *filter;
>

After we make this change, what will set up newsk->sk_net?

That's part of what sk_alloc() was doing for us, and that's
why we need to pass the extra argument.

Subject: Re: [PATCH 1/8] Move the sock_copy() from the header
Posted by davem on Thu, 01 Nov 2007 07:30:02 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 31 Oct 2007 16:42:01 +0300

> The sock_copy() call is not used outside the sock.c file,
> so just move it into a sock.c
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 2/8] Move the get_net() from sock_copy()
Posted by davem on Thu, 01 Nov 2007 07:32:37 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 31 Oct 2007 16:44:40 +0300

> The sock_copy() is supposed to just clone the socket. In a perfect
> world it has to be just memcpy, but we have to handle the security
> mark correctly. All the extra setup must be performed in sk_clone()
> call, so move the get_net() into more proper place.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 3/8] Cleanup the allocation/freeing of the sock object
Posted by davem on Thu, 01 Nov 2007 07:34:08 GMT

View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 31 Oct 2007 16:47:00 +0300

> The sock object is allocated either from the generic cache with
> the kmalloc, or from the proc->slab cache.
>
> Move this logic into an isolated set of helpers and make the
> sk_alloc/sk_free look a bit nicer.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 4/8] Auto-zero the allocated sock object
Posted by davem on Thu, 01 Nov 2007 07:35:00 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 31 Oct 2007 16:48:59 +0300

> We have a __GFP_ZERO flag that allocates a zeroed chunk of memory.
> Use it in the sk_alloc() and avoid a hand-made memset().
>
> This is a temporary patch that will help us in the nearest future :)
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 5/8] Move some core sock setup into sk_prot_alloc
Posted by davem on Thu, 01 Nov 2007 07:36:46 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 31 Oct 2007 16:51:51 +0300

> The security_sk_alloc() and the module_get is a part of the
> object allocations - move it in the proper place.
>
> Note, that since we do not reset the newly allocated sock
> in the sk_alloc() (memset() is removed with the previous
> patch) we can safely do this.
>
> Also fix the error path in sk_prot_alloc() - release the security

> context if needed.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 6/8] Make the sk_clone() lighter
Posted by davem on Thu, 01 Nov 2007 07:38:15 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 31 Oct 2007 16:54:34 +0300

> The sk_prot_alloc() already performs all the stuff needed by the
> sk_clone(). Besides, the sk_prot_alloc() requires almost twice
> less arguments than the sk_alloc() does, so call the sk_prot_alloc()
> saving the stack a bit.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

I reread this patch and now I understand why it's correct.

When zero_it argument is zero, as was the case here, all of
the sk->sk_net processing is not done.

Applied, thanks.

Subject: Re: [PATCH 7/8] Remove bogus zero_it argument from sk_alloc
Posted by davem on Thu, 01 Nov 2007 07:38:59 GMT
View Forum Message <> Reply to Message

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 31 Oct 2007 16:56:10 +0300

> At this point nobody calls the sk_alloc(() with zero_it == 0,
> so remove unneeded checks from it.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 8/8] Forget the zero_it argument of sk_alloc()
Posted by davem on Thu, 01 Nov 2007 07:41:45 GMT

From: Pavel Emelyanov <xemul@openvz.org>
Date: Wed, 31 Oct 2007 16:59:16 +0300

> Finally, the zero_it argument can be completely removed from
> the callers and from the function prototype.
>
> Besides, fix the checkpatch.pl warnings about using the
> assignments inside if-s.
>
> This patch is rather big, and it is a part of the previous one.
> I splitted it wishing to make the patches more readable. Hope
> this particular split helped.
>
> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>

Applied.

Subject: Re: [PATCH 6/8] Make the sk_clone() lighter
Posted by Pavel Emelianov on Thu, 01 Nov 2007 07:44:45 GMT

David Miller wrote:
> From: Pavel Emelyanov <xemul@openvz.org>
> Date: Wed, 31 Oct 2007 16:54:34 +0300
>
>> The sk_prot_alloc() already performs all the stuff needed by the
>> sk_clone(). Besides, the sk_prot_alloc() requires almost twice
>> less arguments than the sk_alloc() does, so call the sk_prot_alloc()
>> saving the stack a bit.
>>
>> Signed-off-by: Pavel Emelyanov <xemul@openvz.org>
>>
>> ---
>>
>> diff --git a/net/core/sock.c b/net/core/sock.c
>> index e7537e4..c032f48 100644
>> --- a/net/core/sock.c
>> +++ b/net/core/sock.c
>> @@ -976,8 +976,9 @@ void sk_free(struct sock *sk)
>>
>> struct sock *sk_clone(const struct sock *sk, const gfp_t priority)
>> {
>> - struct sock *newsk = sk_alloc(sk->sk_net, sk->sk_family, priority, sk->sk_prot, 0);
>> -
>> + struct sock *newsk;

>> +
>> + newsk = sk_prot_alloc(sk->sk_prot, priority, sk->sk_family);
>>   if (newsk != NULL) {
>>   struct sk_filter *filter;
>>
>
> After we make this change, what will set up newsk->sk_net?

This will be done automatically in the sock_copy().

> That's part of what sk_alloc() was doing for us, and that's
> why we need to pass the extra argument.
>

No it wasn't doing it for us, because the sk_net assignment was
done inside the if (zero_it) branch, but zero_it is 0 in this case.

Thanks,
Pavel
