Subject: [PATCH 3/6 net-2.6.25][RAW] Introduce raw_hashinfo structure
Posted by Pavel Emelianov on Fri, 16 Nov 2007 14:12:31 GMT
View Forum Message <> Reply to Message

The ipv4/raw.c and ipv6/raw.c contain many common code (most
of which is proc interface) which can be consolidated.

Most of the places to consolidate deal with the raw sockets
hashtable, so introduce a struct raw_hashinfo which describes
the raw sockets hash.

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

```
diff --git a/include/net/raw.h b/include/net/raw.h
index 7fc3c77..70b27c7 100644
--- a/include/net/raw.h
+++ b/include/net/raw.h
@@ -27,6 +27,13 @@ int raw_local_deliver(struct sk_buff *, int);

 extern int  raw_rcv(struct sock *, struct sk_buff *);

+#define RAW_HTABLE_SIZE MAX_INET_PROTOS
+
+struct raw_hashinfo {
+ rwlock_t lock;
+ struct hlist_head ht[RAW_HTABLE_SIZE];
+};
+
 #ifdef CONFIG_PROC_FS
 extern int  raw_proc_init(void);
 extern void raw_proc_exit(void);
diff --git a/net/ipv4/raw.c b/net/ipv4/raw.c
index 9fda2f9..41a23bc 100644
--- a/net/ipv4/raw.c
+++ b/net/ipv4/raw.c
@@ -80,28 +80,27 @@
 #include <linux/netfilter.h>
 #include <linux/netfilter_ipv4.h>

-#define RAWV4_HTABLE_SIZE MAX_INET_PROTOS
-
-static struct hlist_head raw_v4_htable[RAWV4_HTABLE_SIZE];
-static DEFINE_RWLOCK(raw_v4_lock);
+static struct raw_hashinfo raw_v4_hashinfo = {
+ .lock = __RW_LOCK_UNLOCKED(),
+};
```

```
 static void raw_v4_hash(struct sock *sk)
 {
- struct hlist_head *head = &raw_v4_htable[inet_sk(sk)->num &
-       (RAWV4_HTABLE_SIZE - 1)];
+ struct hlist_head *head = &raw_v4_hashinfo.ht[inet_sk(sk)->num &
+       (RAW_HTABLE_SIZE - 1)];

- write_lock_bh(&raw_v4_lock);
+ write_lock_bh(&raw_v4_hashinfo.lock);
  sk_add_node(sk, head);
  sock_prot_inc_use(sk->sk_prot);
- write_unlock_bh(&raw_v4_lock);
+ write_unlock_bh(&raw_v4_hashinfo.lock);
 }

 static void raw_v4_unhash(struct sock *sk)
 {
- write_lock_bh(&raw_v4_lock);
+ write_lock_bh(&raw_v4_hashinfo.lock);
  if (sk_del_node_init(sk))
   sock_prot_dec_use(sk->sk_prot);
- write_unlock_bh(&raw_v4_lock);
+ write_unlock_bh(&raw_v4_hashinfo.lock);
 }

 static struct sock *__raw_v4_lookup(struct sock *sk, unsigned short num,
@@ -158,8 +157,8 @@ static int raw_v4_input(struct sk_buff *skb, struct iphdr *iph, int hash)
  struct hlist_head *head;
  int delivered = 0;

- read_lock(&raw_v4_lock);
- head = &raw_v4_htable[hash];
+ read_lock(&raw_v4_hashinfo.lock);
+ head = &raw_v4_hashinfo.ht[hash];
  if (hlist_empty(head))
   goto out;
  sk = __raw_v4_lookup(__sk_head(head), iph->protocol,
@@ -180,7 +179,7 @@ static int raw_v4_input(struct sk_buff *skb, struct iphdr *iph, int hash)
         skb->dev->ifindex);
  }
 out:
- read_unlock(&raw_v4_lock);
+ read_unlock(&raw_v4_hashinfo.lock);
  return delivered;
 }

@@ -189,8 +188,8 @@ int raw_local_deliver(struct sk_buff *skb, int protocol)
```

```
  int hash;
  struct sock *raw_sk;

- hash = protocol & (RAWV4_HTABLE_SIZE - 1);
- raw_sk = sk_head(&raw_v4_htable[hash]);
+ hash = protocol & (RAW_HTABLE_SIZE - 1);
+ raw_sk = sk_head(&raw_v4_hashinfo.ht[hash]);

  /* If there maybe a raw socket we must check - if not we
   * don't care less
@@ -262,10 +261,10 @@ void raw_icmp_error(struct sk_buff *skb, int protocol, u32 info)
  struct sock *raw_sk;
  struct iphdr *iph;

- hash = protocol & (RAWV4_HTABLE_SIZE - 1);
+ hash = protocol & (RAW_HTABLE_SIZE - 1);

- read_lock(&raw_v4_lock);
- raw_sk = sk_head(&raw_v4_htable[hash]);
+ read_lock(&raw_v4_hashinfo.lock);
+ raw_sk = sk_head(&raw_v4_hashinfo.ht[hash]);
  if (raw_sk != NULL) {
   iph = (struct iphdr *)skb->data;
   while ((raw_sk = __raw_v4_lookup(raw_sk, protocol, iph->daddr,
@@ -276,7 +275,7 @@ void raw_icmp_error(struct sk_buff *skb, int protocol, u32 info)
    iph = (struct iphdr *)skb->data;
   }
  }
- read_unlock(&raw_v4_lock);
+ read_unlock(&raw_v4_hashinfo.lock);
 }

 static int raw_rcv_skb(struct sock * sk, struct sk_buff * skb)
@@ -842,10 +841,11 @@ static struct sock *raw_get_first(struct seq_file *seq)
  struct sock *sk;
  struct raw_iter_state* state = raw_seq_private(seq);

- for (state->bucket = 0; state->bucket < RAWV4_HTABLE_SIZE; ++state->bucket) {
+ for (state->bucket = 0; state->bucket < RAW_HTABLE_SIZE;
+      ++state->bucket) {
   struct hlist_node *node;

-  sk_for_each(sk, node, &raw_v4_htable[state->bucket])
+  sk_for_each(sk, node, &raw_v4_hashinfo.ht[state->bucket])
    if (sk->sk_family == PF_INET)
     goto found;
  }
@@ -864,8 +864,8 @@ try_again:
```

```
   ;
  } while (sk && sk->sk_family != PF_INET);

- if (!sk && ++state->bucket < RAWV4_HTABLE_SIZE) {
-  sk = sk_head(&raw_v4_htable[state->bucket]);
+ if (!sk && ++state->bucket < RAW_HTABLE_SIZE) {
+  sk = sk_head(&raw_v4_hashinfo.ht[state->bucket]);
    goto try_again;
  }
  return sk;
@@ -883,7 +883,7 @@ static struct sock *raw_get_idx(struct seq_file *seq, loff_t pos)

 static void *raw_seq_start(struct seq_file *seq, loff_t *pos)
 {
- read_lock(&raw_v4_lock);
+ read_lock(&raw_v4_hashinfo.lock);
  return *pos ? raw_get_idx(seq, *pos - 1) : SEQ_START_TOKEN;
 }

@@ -901,7 +901,7 @@ static void *raw_seq_next(struct seq_file *seq, void *v, loff_t *pos)

 static void raw_seq_stop(struct seq_file *seq, void *v)
 {
- read_unlock(&raw_v4_lock);
+ read_unlock(&raw_v4_hashinfo.lock);
 }

 static __inline__ char *get_raw_sock(struct sock *sp, char *tmpbuf, int i)
diff --git a/net/ipv6/raw.c b/net/ipv6/raw.c
index 53f01b4..15c72a6 100644
--- a/net/ipv6/raw.c
+++ b/net/ipv6/raw.c
@@ -54,34 +54,34 @@
 #include <net/mip6.h>
 #endif

+#include <net/raw.h>
 #include <net/rawv6.h>
 #include <net/xfrm.h>

 #include <linux/proc_fs.h>
 #include <linux/seq_file.h>

-#define RAWV6_HTABLE_SIZE MAX_INET_PROTOS
-
-static struct hlist_head raw_v6_htable[RAWV6_HTABLE_SIZE];
-static DEFINE_RWLOCK(raw_v6_lock);
+static struct raw_hashinfo raw_v6_hashinfo = {
```

```
+ .lock = __RW_LOCK_UNLOCKED(),
+};

 static void raw_v6_hash(struct sock *sk)
 {
- struct hlist_head *list = &raw_v6_htable[inet_sk(sk)->num &
-       (RAWV6_HTABLE_SIZE - 1)];
+ struct hlist_head *list = &raw_v6_hashinfo.ht[inet_sk(sk)->num &
+       (RAW_HTABLE_SIZE - 1)];

- write_lock_bh(&raw_v6_lock);
+ write_lock_bh(&raw_v6_hashinfo.lock);
  sk_add_node(sk, list);
  sock_prot_inc_use(sk->sk_prot);
- write_unlock_bh(&raw_v6_lock);
+ write_unlock_bh(&raw_v6_hashinfo.lock);
 }

 static void raw_v6_unhash(struct sock *sk)
 {
- write_lock_bh(&raw_v6_lock);
+ write_lock_bh(&raw_v6_hashinfo.lock);
  if (sk_del_node_init(sk))
   sock_prot_dec_use(sk->sk_prot);
- write_unlock_bh(&raw_v6_lock);
+ write_unlock_bh(&raw_v6_hashinfo.lock);
 }


@@ -180,8 +180,8 @@ static int ipv6_raw_deliver(struct sk_buff *skb, int nexthdr)

  hash = nexthdr & (MAX_INET_PROTOS - 1);

- read_lock(&raw_v6_lock);
- sk = sk_head(&raw_v6_htable[hash]);
+ read_lock(&raw_v6_hashinfo.lock);
+ sk = sk_head(&raw_v6_hashinfo.ht[hash]);

  /*
   * The first socket found will be delivered after
@@ -238,7 +238,7 @@ static int ipv6_raw_deliver(struct sk_buff *skb, int nexthdr)
         IP6CB(skb)->iif);
  }
 out:
- read_unlock(&raw_v6_lock);
+ read_unlock(&raw_v6_hashinfo.lock);
  return delivered;
 }
```

```
@@ -246,7 +246,7 @@ int raw6_local_deliver(struct sk_buff *skb, int nexthdr)
 {
  struct sock *raw_sk;

- raw_sk = sk_head(&raw_v6_htable[nexthdr & (MAX_INET_PROTOS - 1)]);
+ raw_sk = sk_head(&raw_v6_hashinfo.ht[nexthdr & (MAX_INET_PROTOS - 1)]);
  if (raw_sk && !ipv6_raw_deliver(skb, nexthdr))
   raw_sk = NULL;

@@ -368,10 +368,10 @@ void raw6_icmp_error(struct sk_buff *skb, int nexthdr,
  int hash;
  struct in6_addr *saddr, *daddr;

- hash = nexthdr & (RAWV6_HTABLE_SIZE - 1);
+ hash = nexthdr & (RAW_HTABLE_SIZE - 1);

- read_lock(&raw_v6_lock);
- sk = sk_head(&raw_v6_htable[hash]);
+ read_lock(&raw_v6_hashinfo.lock);
+ sk = sk_head(&raw_v6_hashinfo.ht[hash]);
  if (sk != NULL) {
   saddr = &ipv6_hdr(skb)->saddr;
   daddr = &ipv6_hdr(skb)->daddr;
@@ -383,7 +383,7 @@ void raw6_icmp_error(struct sk_buff *skb, int nexthdr,
    sk = sk_next(sk);
   }
  }
- read_unlock(&raw_v6_lock);
+ read_unlock(&raw_v6_hashinfo.lock);
 }

 static inline int rawv6_rcv_skb(struct sock * sk, struct sk_buff * skb)
@@ -1221,8 +1221,9 @@ static struct sock *raw6_get_first(struct seq_file *seq)
  struct hlist_node *node;
  struct raw6_iter_state* state = raw6_seq_private(seq);

- for (state->bucket = 0; state->bucket < RAWV6_HTABLE_SIZE; ++state->bucket)
-  sk_for_each(sk, node, &raw_v6_htable[state->bucket])
+ for (state->bucket = 0; state->bucket < RAW_HTABLE_SIZE;
+    ++state->bucket)
+  sk_for_each(sk, node, &raw_v6_hashinfo.ht[state->bucket])
    if (sk->sk_family == PF_INET6)
     goto out;
  sk = NULL;
@@ -1240,8 +1241,8 @@ try_again:
   ;
  } while (sk && sk->sk_family != PF_INET6);
```

```
- if (!sk && ++state->bucket < RAWV6_HTABLE_SIZE) {
-  sk = sk_head(&raw_v6_htable[state->bucket]);
+ if (!sk && ++state->bucket < RAW_HTABLE_SIZE) {
+  sk = sk_head(&raw_v6_hashinfo.ht[state->bucket]);
   goto try_again;
  }
  return sk;
@@ -1258,7 +1259,7 @@ static struct sock *raw6_get_idx(struct seq_file *seq, loff_t pos)

 static void *raw6_seq_start(struct seq_file *seq, loff_t *pos)
 {
- read_lock(&raw_v6_lock);
+ read_lock(&raw_v6_hashinfo.lock);
  return *pos ? raw6_get_idx(seq, *pos - 1) : SEQ_START_TOKEN;
 }

@@ -1276,7 +1277,7 @@ static void *raw6_seq_next(struct seq_file *seq, void *v, loff_t *pos)

 static void raw6_seq_stop(struct seq_file *seq, void *v)
 {
- read_unlock(&raw_v6_lock);
+ read_unlock(&raw_v6_hashinfo.lock);
 }

 static void raw6_sock_seq_show(struct seq_file *seq, struct sock *sp, int i)
```