Subject: [PATCH 5/5] make netlink user -> kernel interface synchronious
Posted by den on Fri, 05 Oct 2007 14:47:44 GMT
View Forum Message <> Reply to Message

This patch make processing netlink user -> kernel messages synchronious.
This change was inspired by the talk with Alexey Kuznetsov about current
netlink messages processing. He says that he was badly wrong when introduced
asynchronious user -> kernel communication.

The call netlink_unicast is the only path to send message to the kernel
netlink socket. But, unfortunately, it is also used to send data to the
user.

Before this change the user message has been attached to the socket queue
and sk->sk_data_ready was called. The process has been blocked until all
pending messages were processed. The bad thing is that this processing
may occur in the arbitrary process context.

This patch changes nlk->data_ready callback to get 1 skb and force packet
processing right in the netlink_unicast.

Kernel -> user path in netlink_unicast remains untouched.

EINTR processing for in netlink_run_queue was changed. It forces rtnl_lock
drop, but the process remains in the cycle until the message will be fully
processed. So, there is no need to use this kludges now.

Signed-off-by: Denis V. Lunev <den@openvz.org>
Acked-by: Alexey Kuznetsov <kuznet@ms2.inr.ac.ru>

```
--- ./drivers/connector/connector.c.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./drivers/connector/connector.c 2007-10-05 16:33:04.000000000 +0400
@@ -235,18 +235,6 @@ out:
 }

 /*
- * Netlink socket input callback - dequeues the skbs and calls the
- * main netlink receiving function.
- */
-static void cn_input(struct sock *sk, int len)
-{
- struct sk_buff *skb;
-
- while ((skb = skb_dequeue(&sk->sk_receive_queue)) != NULL)
-  cn_rx_skb(skb);
-}
-
-/*
```

```
  * Notification routing.
  *
  * Gets id and checks if there are notification request for it's idx
@@ -442,7 +430,7 @@ static int __devinit cn_init(void)
  struct cn_dev *dev = &cdev;
  int err;

- dev->input = cn_input;
+ dev->input = cn_rx_skb;
  dev->id.idx = cn_idx;
  dev->id.val = cn_val;

--- ./drivers/scsi/scsi_netlink.c.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./drivers/scsi/scsi_netlink.c 2007-10-05 16:49:45.000000000 +0400
@@ -64,7 +64,7 @@ scsi_nl_rcv_msg(struct sk_buff *skb)

   if (nlh->nlmsg_type != SCSI_TRANSPORT_MSG) {
    err = -EBADMSG;
-   goto next_msg;
+   return;
   }

   hdr = NLMSG_DATA(nlh);
@@ -99,27 +99,6 @@ next_msg:


 /**
- * scsi_nl_rcv_msg -
- *    Receive handler for a socket. Extracts a received message buffer from
- *    the socket, and starts message processing.
- *
- * @sk:  socket
- * @len: unused
- *
- **/
-static void
-scsi_nl_rcv(struct sock *sk, int len)
-{
- struct sk_buff *skb;
-
- while ((skb = skb_dequeue(&sk->sk_receive_queue))) {
-  scsi_nl_rcv_msg(skb);
-  kfree_skb(skb);
- }
-}
-
-
-/**
```

```
 * scsi_nl_rcv_event -
 *    Event handler for a netlink socket.
 *
@@ -168,7 +147,7 @@ scsi_netlink_init(void)
  }

  scsi_nl_sock = netlink_kernel_create(&init_net, NETLINK_SCSITRANSPORT,
-    SCSI_NL_GRP_CNT, scsi_nl_rcv, NULL,
+    SCSI_NL_GRP_CNT, scsi_nl_rcv_msg, NULL,
     THIS_MODULE);
  if (!scsi_nl_sock) {
   printk(KERN_ERR "%s: register of recieve handler failed\n",
--- ./drivers/scsi/scsi_transport_iscsi.c.nlk6 2007-10-05 17:51:40.000000000 +0400
+++ ./drivers/scsi/scsi_transport_iscsi.c 2007-10-05 17:53:08.000000000 +0400
@@ -1097,61 +1097,49 @@ iscsi_if_recv_msg(struct sk_buff *skb, s
 }

 /*
- * Get message from skb (based on rtnetlink_rcv_skb).  Each message is
- * processed by iscsi_if_recv_msg.  Malformed skbs with wrong lengths or
- * invalid creds are discarded silently.
+ * Get message from skb.  Each message is processed by iscsi_if_recv_msg.
+ * Malformed skbs with wrong lengths or invalid creds are not processed.
  */
 static void
-iscsi_if_rx(struct sock *sk, int len)
+iscsi_if_rx(struct sk_buff *skb)
 {
- struct sk_buff *skb;
-
  mutex_lock(&rx_queue_mutex);
- while ((skb = skb_dequeue(&sk->sk_receive_queue)) != NULL) {
-  if (NETLINK_CREDS(skb)->uid) {
-   skb_pull(skb, skb->len);
-   goto free_skb;
+ while (skb->len >= NLMSG_SPACE(0)) {
+  int err;
+  uint32_t rlen;
+  struct nlmsghdr *nlh;
+  struct iscsi_uevent *ev;
+
+  nlh = nlmsg_hdr(skb);
+  if (nlh->nlmsg_len < sizeof(*nlh) ||
+      skb->len < nlh->nlmsg_len) {
+   break;
   }

-  while (skb->len >= NLMSG_SPACE(0)) {
```

```
-   int err;
-   uint32_t rlen;
-   struct nlmsghdr *nlh;
-   struct iscsi_uevent *ev;
-
-   nlh = nlmsg_hdr(skb);
-   if (nlh->nlmsg_len < sizeof(*nlh) ||
-       skb->len < nlh->nlmsg_len) {
-    break;
-   }
-
-   ev = NLMSG_DATA(nlh);
-   rlen = NLMSG_ALIGN(nlh->nlmsg_len);
-   if (rlen > skb->len)
-    rlen = skb->len;
-
-   err = iscsi_if_recv_msg(skb, nlh);
-   if (err) {
-    ev->type = ISCSI_KEVENT_IF_ERROR;
-    ev->iferror = err;
-   }
-   do {
-    /*
-     * special case for GET_STATS:
-     * on success - sending reply and stats from
-     * inside of if_recv_msg(),
-     * on error - fall through.
-     */
-    if (ev->type == ISCSI_UEVENT_GET_STATS && !err)
-     break;
-    err = iscsi_if_send_reply(
-     NETLINK_CREDS(skb)->pid, nlh->nlmsg_seq,
-     nlh->nlmsg_type, 0, 0, ev, sizeof(*ev));
-   } while (err < 0 && err != -ECONNREFUSED);
-   skb_pull(skb, rlen);
+  ev = NLMSG_DATA(nlh);
+  rlen = NLMSG_ALIGN(nlh->nlmsg_len);
+  if (rlen > skb->len)
+   rlen = skb->len;
+
+  err = iscsi_if_recv_msg(skb, nlh);
+  if (err) {
+   ev->type = ISCSI_KEVENT_IF_ERROR;
+   ev->iferror = err;
   }
-free_skb:
-  kfree_skb(skb);
+  do {
```

```
+   /*
+    * special case for GET_STATS:
+    * on success - sending reply and stats from
+    * inside of if_recv_msg(),
+    * on error - fall through.
+    */
+   if (ev->type == ISCSI_UEVENT_GET_STATS && !err)
+    break;
+   err = iscsi_if_send_reply(
+    NETLINK_CREDS(skb)->pid, nlh->nlmsg_seq,
+    nlh->nlmsg_type, 0, 0, ev, sizeof(*ev));
+  } while (err < 0 && err != -ECONNREFUSED);
+  skb_pull(skb, rlen);
  }
  mutex_unlock(&rx_queue_mutex);
 }
--- ./fs/ecryptfs/netlink.c.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./fs/ecryptfs/netlink.c 2007-10-05 16:33:04.000000000 +0400
@@ -165,22 +165,10 @@ static int ecryptfs_process_nl_quit(stru
  * it to its desired netlink context element and wake up the process
  * that is waiting for a response.
  */
-static void ecryptfs_receive_nl_message(struct sock *sk, int len)
+static void ecryptfs_receive_nl_message(struct sk_buff *skb)
 {
- struct sk_buff *skb;
  struct nlmsghdr *nlh;
- int rc = 0; /* skb_recv_datagram requires this */

-receive:
- skb = skb_recv_datagram(sk, 0, 0, &rc);
- if (rc == -EINTR)
-  goto receive;
- else if (rc < 0) {
-  ecryptfs_printk(KERN_ERR, "Error occurred while "
-    "receiving eCryptfs netlink message; "
-    "rc = [%d]\n", rc);
-  return;
- }
  nlh = nlmsg_hdr(skb);
  if (!NLMSG_OK(nlh, skb->len)) {
   ecryptfs_printk(KERN_ERR, "Received corrupt netlink "
--- ./include/linux/connector.h.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./include/linux/connector.h 2007-10-05 16:33:04.000000000 +0400
@@ -153,7 +153,7 @@ struct cn_dev {

  u32 seq, groups;
  struct sock *nls;
```

```
- void (*input) (struct sock * sk, int len);
+ void (*input) (struct sk_buff *skb);

  struct cn_queue_dev *cbdev;
 };
--- ./include/linux/netlink.h.nlk6 2007-10-05 16:28:17.000000000 +0400
+++ ./include/linux/netlink.h 2007-10-05 16:33:04.000000000 +0400
@@ -175,7 +175,7 @@ struct netlink_skb_parms

 extern struct sock *netlink_kernel_create(struct net *net,
        int unit,unsigned int groups,
-       void (*input)(struct sock *sk, int len),
+            void (*input)(struct sk_buff *skb),
        struct mutex *cb_mutex,
        struct module *module);
 extern int netlink_change_ngroups(struct sock *sk, unsigned int groups);
--- ./include/net/netlink.h.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./include/net/netlink.h 2007-10-05 16:33:04.000000000 +0400
@@ -220,9 +220,9 @@ struct nl_info {
  u32   pid;
 };

-extern unsigned int netlink_run_queue(struct sock *sk, unsigned int qlen,
-       int (*cb)(struct sk_buff *,
-          struct nlmsghdr *));
+extern int  netlink_rcv_skb(struct sk_buff *skb,
+     int (*cb)(struct sk_buff *,
+        struct nlmsghdr *));
 extern int  nlmsg_notify(struct sock *sk, struct sk_buff *skb,
          u32 pid, unsigned int group, int report,
          gfp_t flags);
--- ./kernel/audit.c.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./kernel/audit.c 2007-10-05 16:50:01.000000000 +0400
@@ -847,18 +847,10 @@ static void audit_receive_skb(struct sk_
 }

 /* Receive messages from netlink socket. */
-static void audit_receive(struct sock *sk, int length)
+static void audit_receive(struct sk_buff  *skb)
 {
- struct sk_buff  *skb;
- unsigned int qlen;
-
  mutex_lock(&audit_cmd_mutex);
-
- for (qlen = skb_queue_len(&sk->sk_receive_queue); qlen; qlen--) {
-  skb = skb_dequeue(&sk->sk_receive_queue);
-  audit_receive_skb(skb);
```

```
-  kfree_skb(skb);
- }
+ audit_receive_skb(skb);
  mutex_unlock(&audit_cmd_mutex);
 }

--- ./net/core/rtnetlink.c.nlk6 2007-10-05 16:28:09.000000000 +0400
+++ ./net/core/rtnetlink.c 2007-10-05 16:50:29.000000000 +0400
@@ -1312,15 +1312,11 @@ static int rtnetlink_rcv_msg(struct sk_b
  return doit(skb, nlh, (void *)&rta_buf[0]);
 }

-static void rtnetlink_rcv(struct sock *sk, int len)
+static void rtnetlink_rcv(struct sk_buff *skb)
 {
- unsigned int qlen = 0;
-
- do {
-  rtnl_lock();
-  qlen = netlink_run_queue(sk, qlen, &rtnetlink_rcv_msg);
-  rtnl_unlock();
- } while (qlen);
+ rtnl_lock();
+ netlink_rcv_skb(skb, &rtnetlink_rcv_msg);
+ rtnl_unlock();
 }

 static int rtnetlink_event(struct notifier_block *this, unsigned long event, void *ptr)
--- ./net/decnet/netfilter/dn_rtmsg.c.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./net/decnet/netfilter/dn_rtmsg.c 2007-10-05 16:50:45.000000000 +0400
@@ -115,17 +115,6 @@ static inline void dnrmg_receive_user_sk
  RCV_SKB_FAIL(-EINVAL);
 }

-static void dnrmg_receive_user_sk(struct sock *sk, int len)
-{
- struct sk_buff *skb;
- unsigned int qlen = skb_queue_len(&sk->sk_receive_queue);
-
- for (; qlen && (skb = skb_dequeue(&sk->sk_receive_queue)); qlen--) {
-  dnrmg_receive_user_skb(skb);
-  kfree_skb(skb);
- }
-}
-
 static struct nf_hook_ops dnrmg_ops = {
  .hook  = dnrmg_hook,
  .pf  = PF_DECnet,
```

```
@@ -139,7 +128,8 @@ static int __init dn_rtmsg_init(void)

  dnrmg = netlink_kernel_create(&init_net,
          NETLINK_DNRTMSG, DNRNG_NLGRP_MAX,
-         dnrmg_receive_user_sk, NULL, THIS_MODULE);
+         dnrmg_receive_user_skb,
+         NULL, THIS_MODULE);
  if (dnrmg == NULL) {
   printk(KERN_ERR "dn_rtmsg: Cannot create netlink socket");
   return -ENOMEM;
--- ./net/ipv4/fib_frontend.c.nlk6 2007-10-05 16:33:04.000000000 +0400
+++ ./net/ipv4/fib_frontend.c 2007-10-05 16:33:14.000000000 +0400
@@ -62,6 +62,9 @@ static struct hlist_head fib_table_hash[
 #define FIB_TABLE_HASHSZ 256
 static struct hlist_head fib_table_hash[FIB_TABLE_HASHSZ];

+static struct sock *fibnl = NULL;
+
+
 struct fib_table *fib_new_table(u32 id)
 {
  struct fib_table *tb;
@@ -806,13 +809,13 @@ static void nl_fib_input(struct sk_buff
  pid = NETLINK_CB(skb).pid;       /* pid of sending process */
  NETLINK_CB(skb).pid = 0;         /* from kernel */
  NETLINK_CB(skb).dst_group = 0;  /* unicast */
- netlink_unicast(sk, skb, pid, MSG_DONTWAIT);
+ netlink_unicast(fibnl, skb, pid, MSG_DONTWAIT);
 }

 static void nl_fib_lookup_init(void)
 {
- netlink_kernel_create(&init_net, NETLINK_FIB_LOOKUP, 0, nl_fib_input,
-    NULL, THIS_MODULE);
+ fibnl = netlink_kernel_create(&init_net, NETLINK_FIB_LOOKUP, 0,
+         nl_fib_input, NULL, THIS_MODULE);
 }

 static void fib_disable_ip(struct net_device *dev, int force)
--- ./net/ipv4/inet_diag.c.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./net/ipv4/inet_diag.c 2007-10-05 16:50:56.000000000 +0400
@@ -839,15 +839,11 @@ static int inet_diag_rcv_msg(struct sk_b

 static DEFINE_MUTEX(inet_diag_mutex);

-static void inet_diag_rcv(struct sock *sk, int len)
+static void inet_diag_rcv(struct sk_buff *skb)
 {
```

```
- unsigned int qlen = 0;
-
- do {
-  mutex_lock(&inet_diag_mutex);
-  qlen = netlink_run_queue(sk, qlen, &inet_diag_rcv_msg);
-  mutex_unlock(&inet_diag_mutex);
- } while (qlen);
+ mutex_lock(&inet_diag_mutex);
+ netlink_rcv_skb(skb, &inet_diag_rcv_msg);
+ mutex_unlock(&inet_diag_mutex);
 }

 static DEFINE_SPINLOCK(inet_diag_register_lock);
--- ./net/ipv4/netfilter/ip_queue.c.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./net/ipv4/netfilter/ip_queue.c 2007-10-05 16:51:04.000000000 +0400
@@ -475,7 +475,7 @@ ipq_dev_drop(int ifindex)
 #define RCV_SKB_FAIL(err) do { netlink_ack(skb, nlh, (err)); return; } while (0)

 static inline void
-ipq_rcv_skb(struct sk_buff *skb)
+__ipq_rcv_skb(struct sk_buff *skb)
 {
  int status, type, pid, flags, nlmsglen, skblen;
  struct nlmsghdr *nlh;
@@ -533,19 +533,10 @@ ipq_rcv_skb(struct sk_buff *skb)
 }

 static void
-ipq_rcv_sk(struct sock *sk, int len)
+ipq_rcv_skb(struct sk_buff *skb)
 {
- struct sk_buff *skb;
- unsigned int qlen;
-
  mutex_lock(&ipqnl_mutex);
-
- for (qlen = skb_queue_len(&sk->sk_receive_queue); qlen; qlen--) {
-  skb = skb_dequeue(&sk->sk_receive_queue);
-  ipq_rcv_skb(skb);
-  kfree_skb(skb);
- }
-
+ __ipq_rcv_skb(skb);
  mutex_unlock(&ipqnl_mutex);
 }

@@ -670,7 +661,7 @@ static int __init ip_queue_init(void)
```

```
  netlink_register_notifier(&ipq_nl_notifier);
  ipqnl = netlink_kernel_create(&init_net, NETLINK_FIREWALL, 0,
-          ipq_rcv_sk, NULL, THIS_MODULE);
+          ipq_rcv_skb, NULL, THIS_MODULE);
  if (ipqnl == NULL) {
   printk(KERN_ERR "ip_queue: failed to create netlink socket\n");
   goto cleanup_netlink_notifier;
--- ./net/ipv6/netfilter/ip6_queue.c.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./net/ipv6/netfilter/ip6_queue.c 2007-10-05 16:51:11.000000000 +0400
@@ -464,7 +464,7 @@ ipq_dev_drop(int ifindex)
 #define RCV_SKB_FAIL(err) do { netlink_ack(skb, nlh, (err)); return; } while (0)

 static inline void
-ipq_rcv_skb(struct sk_buff *skb)
+__ipq_rcv_skb(struct sk_buff *skb)
 {
  int status, type, pid, flags, nlmsglen, skblen;
  struct nlmsghdr *nlh;
@@ -522,19 +522,10 @@ ipq_rcv_skb(struct sk_buff *skb)
 }

 static void
-ipq_rcv_sk(struct sock *sk, int len)
+ipq_rcv_skb(struct sk_buff *skb)
 {
- struct sk_buff *skb;
- unsigned int qlen;
-
  mutex_lock(&ipqnl_mutex);
-
- for (qlen = skb_queue_len(&sk->sk_receive_queue); qlen; qlen--) {
-  skb = skb_dequeue(&sk->sk_receive_queue);
-  ipq_rcv_skb(skb);
-  kfree_skb(skb);
- }
-
+ __ipq_rcv_skb(skb);
  mutex_unlock(&ipqnl_mutex);
 }

@@ -658,8 +649,8 @@ static int __init ip6_queue_init(void)
  struct proc_dir_entry *proc;

  netlink_register_notifier(&ipq_nl_notifier);
- ipqnl = netlink_kernel_create(&init_net, NETLINK_IP6_FW, 0, ipq_rcv_sk,
-     NULL, THIS_MODULE);
+ ipqnl = netlink_kernel_create(&init_net, NETLINK_IP6_FW, 0,
+                ipq_rcv_skb, NULL, THIS_MODULE);
```

```
  if (ipqnl == NULL) {
   printk(KERN_ERR "ip6_queue: failed to create netlink socket\n");
   goto cleanup_netlink_notifier;
--- ./net/netfilter/nfnetlink.c.nlk6 2007-10-05 16:28:13.000000000 +0400
+++ ./net/netfilter/nfnetlink.c 2007-10-05 16:51:28.000000000 +0400
@@ -169,15 +169,11 @@ static int nfnetlink_rcv_msg(struct sk_b
  }
 }

-static void nfnetlink_rcv(struct sock *sk, int len)
+static void nfnetlink_rcv(struct sk_buff *skb)
 {
- unsigned int qlen = 0;
-
- do {
-  nfnl_lock();
-  qlen = netlink_run_queue(sk, qlen, nfnetlink_rcv_msg);
-  nfnl_unlock();
- } while (qlen);
+ nfnl_lock();
+ netlink_rcv_skb(skb, &nfnetlink_rcv_msg);
+ nfnl_unlock();
 }

 static void __exit nfnetlink_exit(void)
--- ./net/netlink/af_netlink.c.nlk6 2007-10-05 16:29:35.000000000 +0400
+++ ./net/netlink/af_netlink.c 2007-10-05 18:07:41.000000000 +0400
@@ -80,7 +80,7 @@ struct netlink_sock {
  struct netlink_callback *cb;
  struct mutex  *cb_mutex;
  struct mutex  cb_def_mutex;
- void   (*data_ready)(struct sock *sk, int bytes);
+ void   (*netlink_rcv)(struct sk_buff *skb);
  struct module  *module;
 };

@@ -127,7 +127,6 @@ static DECLARE_WAIT_QUEUE_HEAD(nl_table_

 static int netlink_dump(struct sock *sk);
 static void netlink_destroy_callback(struct netlink_callback *cb);
-static void netlink_queue_skip(struct nlmsghdr *nlh, struct sk_buff *skb);

 static DEFINE_RWLOCK(nl_table_lock);
 static atomic_t nl_table_users = ATOMIC_INIT(0);
@@ -711,21 +710,17 @@ static void netlink_overrun(struct sock

 static struct sock *netlink_getsockbypid(struct sock *ssk, u32 pid)
 {
```

```
- int protocol = ssk->sk_protocol;
- struct net *net;
  struct sock *sock;
  struct netlink_sock *nlk;

- net = ssk->sk_net;
- sock = netlink_lookup(net, protocol, pid);
+ sock = netlink_lookup(ssk->sk_net, ssk->sk_protocol, pid);
  if (!sock)
   return ERR_PTR(-ECONNREFUSED);

  /* Don't bother queuing skb if kernel socket has no input function */
  nlk = nlk_sk(sock);
- if ((netlink_is_kernel(sock) && !nlk->data_ready) ||
-     (sock->sk_state == NETLINK_CONNECTED &&
-      nlk->dst_pid != nlk_sk(ssk)->pid)) {
+ if (sock->sk_state == NETLINK_CONNECTED &&
+     nlk->dst_pid != nlk_sk(ssk)->pid) {
   sock_put(sock);
   return ERR_PTR(-ECONNREFUSED);
  }
@@ -838,7 +833,34 @@ static inline struct sk_buff *netlink_tr
  return skb;
 }

-int netlink_unicast(struct sock *ssk, struct sk_buff *skb, u32 pid, int nonblock)
+static inline void netlink_rcv_wake(struct sock *sk)
+{
+ struct netlink_sock *nlk = nlk_sk(sk);
+
+ if (skb_queue_empty(&sk->sk_receive_queue))
+  clear_bit(0, &nlk->state);
+ if (!test_bit(0, &nlk->state))
+  wake_up_interruptible(&nlk->wait);
+}
+
+static inline int netlink_unicast_kernel(struct sock *sk, struct sk_buff *skb)
+{
+ int ret;
+ struct netlink_sock *nlk = nlk_sk(sk);
+
+ ret = -ECONNREFUSED;
+ if (nlk->netlink_rcv != NULL) {
+  ret = skb->len;
+  skb_set_owner_r(skb, sk);
+  nlk->netlink_rcv(skb);
+ }
+ kfree_skb(skb);
```

```
+ sock_put(sk);
+ return ret;
+}
+
+int netlink_unicast(struct sock *ssk, struct sk_buff *skb,
+       u32 pid, int nonblock)
 {
  struct sock *sk;
  int err;
@@ -853,6 +875,9 @@ retry:
    kfree_skb(skb);
    return PTR_ERR(sk);
  }
+ if (netlink_is_kernel(sk))
+  return netlink_unicast_kernel(sk, skb);
+
  err = netlink_attachskb(sk, skb, nonblock, timeo, ssk);
  if (err == 1)
    goto retry;
@@ -1152,16 +1177,6 @@ static void netlink_cmsg_recv_pktinfo(st
  put_cmsg(msg, SOL_NETLINK, NETLINK_PKTINFO, sizeof(info), &info);
 }

-static inline void netlink_rcv_wake(struct sock *sk)
-{
- struct netlink_sock *nlk = nlk_sk(sk);
-
- if (skb_queue_empty(&sk->sk_receive_queue))
-  clear_bit(0, &nlk->state);
- if (!test_bit(0, &nlk->state))
-  wake_up_interruptible(&nlk->wait);
-}
-
 static int netlink_sendmsg(struct kiocb *kiocb, struct socket *sock,
        struct msghdr *msg, size_t len)
 {
@@ -1309,11 +1324,7 @@ out:

 static void netlink_data_ready(struct sock *sk, int len)
 {
- struct netlink_sock *nlk = nlk_sk(sk);
-
- if (nlk->data_ready)
-  nlk->data_ready(sk, len);
- netlink_rcv_wake(sk);
+ BUG();
 }
```

```
 /*
@@ -1324,7 +1335,7 @@ static void netlink_data_ready(struct so

 struct sock *
 netlink_kernel_create(struct net *net, int unit, unsigned int groups,
-        void (*input)(struct sock *sk, int len),
+        void (*input)(struct sk_buff *skb),
         struct mutex *cb_mutex, struct module *module)
 {
  struct socket *sock;
@@ -1353,7 +1364,7 @@ netlink_kernel_create(struct net *net, i
  sk = sock->sk;
  sk->sk_data_ready = netlink_data_ready;
  if (input)
-  nlk_sk(sk)->data_ready = input;
+  nlk_sk(sk)->netlink_rcv = input;

  if (netlink_insert(sk, net, 0))
   goto out_sock_release;
@@ -1553,12 +1564,7 @@ int netlink_dump_start(struct sock *ssk,

  netlink_dump(sk);
  sock_put(sk);
-
- /* We successfully started a dump, by returning -EINTR we
-  * signal the queue mangement to interrupt processing of
-  * any netlink messages so userspace gets a chance to read
-  * the results. */
- return -EINTR;
+ return 0;
 }

 void netlink_ack(struct sk_buff *in_skb, struct nlmsghdr *nlh, int err)
@@ -1595,13 +1601,15 @@ void netlink_ack(struct sk_buff *in_skb,
  netlink_unicast(in_skb->sk, skb, NETLINK_CB(in_skb).pid, MSG_DONTWAIT);
 }

-static int netlink_rcv_skb(struct sk_buff *skb, int (*cb)(struct sk_buff *,
+int netlink_rcv_skb(struct sk_buff *skb, int (*cb)(struct sk_buff *,
           struct nlmsghdr *))
 {
  struct nlmsghdr *nlh;
  int err;

  while (skb->len >= nlmsg_total_size(0)) {
+  int msglen;
+
   nlh = nlmsg_hdr(skb);
```

```
   err = 0;

@@ -1617,86 +1625,20 @@ static int netlink_rcv_skb(struct sk_buf
    goto skip;

   err = cb(skb, nlh);
-  if (err == -EINTR) {
-   /* Not an error, but we interrupt processing */
-   netlink_queue_skip(nlh, skb);
-   return err;
-  }
 skip:
   if (nlh->nlmsg_flags & NLM_F_ACK || err)
    netlink_ack(skb, nlh, err);

-  netlink_queue_skip(nlh, skb);
+         msglen = NLMSG_ALIGN(nlh->nlmsg_len);
+  if (msglen > skb->len)
+   msglen = skb->len;
+  skb_pull(skb, msglen);
  }

  return 0;
 }

 /**
- * nelink_run_queue - Process netlink receive queue.
- * @sk: Netlink socket containing the queue
- * @qlen: Initial queue length
- * @cb: Callback function invoked for each netlink message found
- *
- * Processes as much as there was in the queue upon entry and invokes
- * a callback function for each netlink message found. The callback
- * function may refuse a message by returning a negative error code
- * but setting the error pointer to 0 in which case this function
- * returns with a qlen != 0.
- *
- * qlen must be initialized to 0 before the initial entry, afterwards
- * the function may be called repeatedly until the returned qlen is 0.
- *
- * The callback function may return -EINTR to signal that processing
- * of netlink messages shall be interrupted. In this case the message
- * currently being processed will NOT be requeued onto the receive
- * queue.
- */
-unsigned int netlink_run_queue(struct sock *sk, unsigned int qlen,
-         int (*cb)(struct sk_buff *, struct nlmsghdr *))
-{
```

```
- struct sk_buff *skb;
-
- if (!qlen || qlen > skb_queue_len(&sk->sk_receive_queue))
-  qlen = skb_queue_len(&sk->sk_receive_queue);
-
- for (; qlen; qlen--) {
-  skb = skb_dequeue(&sk->sk_receive_queue);
-  if (netlink_rcv_skb(skb, cb)) {
-   if (skb->len)
-    skb_queue_head(&sk->sk_receive_queue, skb);
-   else {
-    kfree_skb(skb);
-    qlen--;
-   }
-   break;
-  }
-
-  kfree_skb(skb);
- }
-
- return qlen;
-}
-
-/**
- * netlink_queue_skip - Skip netlink message while processing queue.
- * @nlh: Netlink message to be skipped
- * @skb: Socket buffer containing the netlink messages.
- *
- * Pulls the given netlink message off the socket buffer so the next
- * call to netlink_queue_run() will not reconsider the message.
- */
-static void netlink_queue_skip(struct nlmsghdr *nlh, struct sk_buff *skb)
-{
- int msglen = NLMSG_ALIGN(nlh->nlmsg_len);
-
- if (msglen > skb->len)
-  msglen = skb->len;
-
- skb_pull(skb, msglen);
-}
-
-/**
  * nlmsg_notify - send a notification netlink message
  * @sk: netlink socket to use
  * @skb: notification message
@@ -2008,7 +1950,7 @@ panic:
 core_initcall(netlink_proto_init);
```

```
 EXPORT_SYMBOL(netlink_ack);
-EXPORT_SYMBOL(netlink_run_queue);
+EXPORT_SYMBOL(netlink_rcv_skb);
 EXPORT_SYMBOL(netlink_broadcast);
 EXPORT_SYMBOL(netlink_dump_start);
 EXPORT_SYMBOL(netlink_kernel_create);
--- ./net/netlink/genetlink.c.nlk6 2007-10-05 16:28:13.000000000 +0400
+++ ./net/netlink/genetlink.c 2007-10-05 16:52:21.000000000 +0400
@@ -470,15 +470,11 @@ static int genl_rcv_msg(struct sk_buff *
  return ops->doit(skb, &info);
 }

-static void genl_rcv(struct sock *sk, int len)
+static void genl_rcv(struct sk_buff *skb)
 {
- unsigned int qlen = 0;
-
- do {
-  genl_lock();
-  qlen = netlink_run_queue(sk, qlen, genl_rcv_msg);
-  genl_unlock();
- } while (qlen && genl_sock && genl_sock->sk_receive_queue.qlen);
+ genl_lock();
+ netlink_rcv_skb(skb, &genl_rcv_msg);
+ genl_unlock();
 }

 /**************************************************************************
--- ./net/xfrm/xfrm_user.c.nlk6 2007-10-05 16:27:16.000000000 +0400
+++ ./net/xfrm/xfrm_user.c 2007-10-05 16:52:31.000000000 +0400
@@ -1881,16 +1881,11 @@ static int xfrm_user_rcv_msg(struct sk_b
  return link->doit(skb, nlh, attrs);
 }

-static void xfrm_netlink_rcv(struct sock *sk, int len)
+static void xfrm_netlink_rcv(struct sk_buff *skb)
 {
- unsigned int qlen = 0;
-
- do {
-  mutex_lock(&xfrm_cfg_mutex);
-  qlen = netlink_run_queue(sk, qlen, &xfrm_user_rcv_msg);
-  mutex_unlock(&xfrm_cfg_mutex);
-
- } while (qlen);
+ mutex_lock(&xfrm_cfg_mutex);
+ netlink_rcv_skb(skb, &xfrm_user_rcv_msg);
+ mutex_unlock(&xfrm_cfg_mutex);
```

}

static inline size_t xfrm_expire_msgsize(void)

Subject: Re: [PATCH 5/5] make netlink user -> kernel interface synchronious
Posted by Patrick McHardy on Fri, 05 Oct 2007 16:24:27 GMT
View Forum Message <> Reply to Message

Denis V. Lunev wrote:
> This patch make processing netlink user -> kernel messages synchronious.
> This change was inspired by the talk with Alexey Kuznetsov about current
> netlink messages processing. He says that he was badly wrong when introduced
> asynchronious user -> kernel communication.
>
> The call netlink_unicast is the only path to send message to the kernel
> netlink socket. But, unfortunately, it is also used to send data to the
> user.
>
> Before this change the user message has been attached to the socket queue
> and sk->sk_data_ready was called. The process has been blocked until all
> pending messages were processed. The bad thing is that this processing
> may occur in the arbitrary process context.
>
> This patch changes nlk->data_ready callback to get 1 skb and force packet
> processing right in the netlink_unicast.

I guess the process credential stuff in netlink_skb_params can now
be removed as well.

Subject: Re: Re: [PATCH 5/5] make netlink user -> kernel interface synchronious
Posted by dlunev on Fri, 05 Oct 2007 16:55:24 GMT
View Forum Message <> Reply to Message

Patrick McHardy wrote:
> Denis V. Lunev wrote:
>> This patch make processing netlink user -> kernel messages synchronious.
>> This change was inspired by the talk with Alexey Kuznetsov about current
>> netlink messages processing. He says that he was badly wrong when introduced
>> asynchronious user -> kernel communication.
>>
>> The call netlink_unicast is the only path to send message to the kernel
>> netlink socket. But, unfortunately, it is also used to send data to the
>> user.
>>

>> Before this change the user message has been attached to the socket queue
>> and sk->sk_data_ready was called. The process has been blocked until all
>> pending messages were processed. The bad thing is that this processing
>> may occur in the arbitrary process context.
>>
>> This patch changes nlk->data_ready callback to get 1 skb and force packet
>> processing right in the netlink_unicast.
>
>
> I guess the process credential stuff in netlink_skb_params can now
> be removed as well.

Yes, but I think the patch should be tested by more people before.

Regards,
Den

Subject: Re: [PATCH 5/5] make netlink user -> kernel interface synchronious
Posted by davem on Thu, 11 Oct 2007 04:15:38 GMT
View Forum Message <> Reply to Message

From: "Denis V. Lunev" <den@openvz.org>
Date: Fri, 5 Oct 2007 18:48:44 +0400

> This patch make processing netlink user -> kernel messages synchronious.
> This change was inspired by the talk with Alexey Kuznetsov about current
> netlink messages processing. He says that he was badly wrong when introduced
> asynchronious user -> kernel communication.
>
> The call netlink_unicast is the only path to send message to the kernel
> netlink socket. But, unfortunately, it is also used to send data to the
> user.
>
> Before this change the user message has been attached to the socket queue
> and sk->sk_data_ready was called. The process has been blocked until all
> pending messages were processed. The bad thing is that this processing
> may occur in the arbitrary process context.
>
> This patch changes nlk->data_ready callback to get 1 skb and force packet
> processing right in the netlink_unicast.
>
> Kernel -> user path in netlink_unicast remains untouched.
>
> EINTR processing for in netlink_run_queue was changed. It forces rtnl_lock
> drop, but the process remains in the cycle until the message will be fully
> processed. So, there is no need to use this kludges now.
>

> Signed-off-by: Denis V. Lunev <den@openvz.org>
> Acked-by: Alexey Kuznetsov <kuznet@ms2.inr.ac.ru>

Applied.
