Subject: [PATCH RFC 31/31] net: Add etun driver
Posted by ebiederm on Thu, 25 Jan 2007 19:00:33 GMT
View Forum Message <> Reply to Message

From: Eric W. Biederman <ebiederm@xmission.com> - unquoted

etun is a simple two headed tunnel driver that at the link layer
looks like ethernet. It's target audience is communicating
between network namespaces but it is general enough it may
have other uses as well.

Signed-off-by: Eric W. Biederman <ebiederm@xmission.com>
---
drivers/net/Kconfig | 14 ++
drivers/net/Makefile | 1 +
drivers/net/etun.c | 470 ++
3 files changed, 485 insertions(+), 0 deletions(-)

diff --git a/drivers/net/Kconfig b/drivers/net/Kconfig
index 8aa8dd0..969d3df 100644
--- a/drivers/net/Kconfig
+++ b/drivers/net/Kconfig
@@ -119,6 +119,20 @@ config TUN

If you don't know what to use this for, you don't need it.

+config ETUN
+ tristate "Ethernet tunnel device driver support"
+ depends on SYSFS
+ ---help---
+ ETUN provices a pair of network devices that can be used for
+ configuring interesting topolgies. What one devices transmits
+ the other receives and vice versa. The link level framing
+ is ethernet for wide compatibility with network stacks.
+
+ To compile this driver as a module, choose M here: the module
+ will be called etun.
+
+ If you don't know what to use this for, you don't need it.
+
config NET_SB1000
tristate "General Instruments Surfboard 1000"
depends on PNP
diff --git a/drivers/net/Makefile b/drivers/net/Makefile
index 4c0d4e5..396af4f 100644
--- a/drivers/net/Makefile
+++ b/drivers/net/Makefile
@@ -185,6 +185,7 @@ obj-$(CONFIG_MACSONIC) += macsonic.o

```
 obj-$(CONFIG_MACMACE) += macmace.o
 obj-$(CONFIG_MAC89x0) += mac89x0.o
 obj-$(CONFIG_TUN) += tun.o
+obj-$(CONFIG_ETUN) += etun.o
 obj-$(CONFIG_NET_NETX) += netx-eth.o
 obj-$(CONFIG_DL2K) += dl2k.o
 obj-$(CONFIG_R8169) += r8169.o
diff --git a/drivers/net/etun.c b/drivers/net/etun.c
new file mode 100644
index 0000000..1dd8cd8
--- /dev/null
+++ b/drivers/net/etun.c
@@ -0,0 +1,470 @@
+/*
+ *  ETUN - Universal ETUN device driver.
+ *  Copyright (C) 2006 Linux Networx
+ *
+ */
+
+#define DRV_NAME "etun"
+#define DRV_VERSION "1.0"
+#define DRV_DESCRIPTION "Ethernet pseudo tunnel device driver"
+#define DRV_COPYRIGHT "(C) 2007 Linux Networx"
+
+#include <linux/module.h>
+#include <linux/kernel.h>
+#include <linux/list.h>
+#include <linux/spinlock.h>
+#include <linux/skbuff.h>
+#include <linux/netdevice.h>
+#include <linux/etherdevice.h>
+#include <linux/ethtool.h>
+#include <linux/rtnetlink.h>
+#include <linux/if.h>
+#include <linux/if_ether.h>
+#include <linux/ctype.h>
+#include <net/net_namespace.h>
+#include <net/dst.h>
+
+
+/* Device cheksum strategy.
+ *
+ * etun is designed to a be a pair of virutal devices
+ * connecting two network stack instances.
+ *
+ * Typically it will either be used with ethernet bridging or
+ * it will be used to route packets between the two stacks.
+ *
```

```
+ * The only checksum offloading I can do is to completely
+ * skip the checksumming step all together.
+ *
+ * When used for ethernet bridging I don't believe any
+ * checksum off loading is safe.
+ * - If my source is an external interface the checksum may be
+ *   invalid so I don't want to report I have already checked it.
+ * - If my destination is an external interface I don't want to put
+ *   a packet on the wire with someone computing the checksum.
+ *
+ * When used for routing between two stacks checksums should
+ * be as unnecessary as they are on the loopback device.
+ *
+ * So by default I am safe and disable checksumming and
+ * other advanced features like SG and TSO.
+ *
+ * However because I think these features could be useful
+ * I provide the ethtool functions to and enable/disable
+ * them at runtime.
+ *
+ * If you think you can correctly enable these go ahead.
+ * For checksums both the transmitter and the receiver must
+ * agree before the are actually disabled.
+ */
+
+#define ETUN_NUM_STATS 1
+static struct {
+ const char string[ETH_GSTRING_LEN];
+} ethtool_stats_keys[ETUN_NUM_STATS] = {
+ { "partner_ifindex" },
+};
+
+struct etun_info {
+ struct net_device  *rx_dev;
+ unsigned  ip_summed;
+ struct net_device_stats stats;
+ struct list_head list;
+ struct net_device *dev;
+};
+
+/*
+ * I have to hold the rtnl_lock during device delete.
+ * So I use the rtnl_lock to protect my list manipulations
+ * as well.  Crude but simple.
+ */
+static LIST_HEAD(etun_list);
+
+/*
```

```
+ * The higher levels take care of making this non-reentrant (it's
+ * called with bh's disabled).
+ */
+static int etun_xmit(struct sk_buff *skb, struct net_device *tx_dev)
+{
+ struct etun_info *tx_info = tx_dev->priv;
+ struct net_device *rx_dev = tx_info->rx_dev;
+ struct etun_info *rx_info = rx_dev->priv;
+
+ tx_info->stats.tx_packets++;
+ tx_info->stats.tx_bytes += skb->len;
+
+ /* Drop the skb state that was needed to get here */
+ skb_orphan(skb);
+ if (skb->dst)
+  skb->dst = dst_pop(skb->dst); /* Allow for smart routing */
+
+ /* Switch to the receiving device */
+ skb->pkt_type = PACKET_HOST;
+ skb->protocol = eth_type_trans(skb, rx_dev);
+ skb->dev = rx_dev;
+ skb->ip_summed = CHECKSUM_NONE;
+
+ /* If both halves agree no checksum is needed */
+ if (tx_dev->features & NETIF_F_NO_CSUM)
+  skb->ip_summed = rx_info->ip_summed;
+
+ rx_dev->last_rx = jiffies;
+ rx_info->stats.rx_packets++;
+ rx_info->stats.rx_bytes += skb->len;
+ netif_rx(skb);
+
+ return 0;
+}
+
+static struct net_device_stats *etun_get_stats(struct net_device *dev)
+{
+ struct etun_info *info = dev->priv;
+ return &info->stats;
+}
+
+/* ethtool interface */
+static int etun_get_settings(struct net_device *dev, struct ethtool_cmd *cmd)
+{
+ cmd->supported  = 0;
+ cmd->advertising = 0;
+ cmd->speed  = SPEED_10000; /* Memory is fast! */
+ cmd->duplex  = DUPLEX_FULL;
```

```
+ cmd->port  = PORT_TP;
+ cmd->phy_address = 0;
+ cmd->transceiver = XCVR_INTERNAL;
+ cmd->autoneg  = AUTONEG_DISABLE;
+ cmd->maxtxpkt  = 0;
+ cmd->maxrxpkt  = 0;
+ return 0;
+}
+
+static void etun_get_drvinfo(struct net_device *dev, struct ethtool_drvinfo *info)
+{
+ strcpy(info->driver, DRV_NAME);
+ strcpy(info->version, DRV_VERSION);
+ strcpy(info->fw_version, "N/A");
+}
+
+static void etun_get_strings(struct net_device *dev, u32 stringset, u8 *buf)
+{
+ switch(stringset) {
+ case ETH_SS_STATS:
+  memcpy(buf, &ethtool_stats_keys, sizeof(ethtool_stats_keys));
+  break;
+ case ETH_SS_TEST:
+ default:
+  break;
+ }
+}
+
+static int etun_get_stats_count(struct net_device *dev)
+{
+ return ETUN_NUM_STATS;
+}
+
+static void etun_get_ethtool_stats(struct net_device *dev,
+ struct ethtool_stats *stats, u64 *data)
+{
+ struct etun_info *info = dev->priv;
+
+ data[0] = info->rx_dev->ifindex;
+}
+
+static u32 etun_get_rx_csum(struct net_device *dev)
+{
+ struct etun_info *info = dev->priv;
+ return info->ip_summed == CHECKSUM_UNNECESSARY;
+}
+
+static int etun_set_rx_csum(struct net_device *dev, u32 data)
```

```
+{
+ struct etun_info *info = dev->priv;
+
+ info->ip_summed = data ? CHECKSUM_UNNECESSARY : CHECKSUM_NONE;
+
+ return 0;
+}
+
+static u32 etun_get_tx_csum(struct net_device *dev)
+{
+ return (dev->features & NETIF_F_NO_CSUM) != 0;
+}
+
+static int etun_set_tx_csum(struct net_device *dev, u32 data)
+{
+ dev->features &= NETIF_F_NO_CSUM;
+ if (data)
+  dev->features |= NETIF_F_NO_CSUM;
+
+ return 0;
+}
+
+static struct ethtool_ops etun_ethtool_ops = {
+ .get_settings  = etun_get_settings,
+ .get_drvinfo  = etun_get_drvinfo,
+ .get_link  = ethtool_op_get_link,
+ .get_rx_csum  = etun_get_rx_csum,
+ .set_rx_csum  = etun_set_rx_csum,
+ .get_tx_csum  = etun_get_tx_csum,
+ .set_tx_csum  = etun_set_tx_csum,
+ .get_sg   = ethtool_op_get_sg,
+ .set_sg   = ethtool_op_set_sg,
+#if 0 /* Does just setting the bit successfuly emulate tso? */
+ .get_tso  = ethtool_op_get_tso,
+ .set_tso  = ethtool_op_set_tso,
+#endif
+ .get_strings  = etun_get_strings,
+ .get_stats_count = etun_get_stats_count,
+ .get_ethtool_stats = etun_get_ethtool_stats,
+ .get_perm_addr  = ethtool_op_get_perm_addr,
+};
+
+static int etun_open(struct net_device *tx_dev)
+{
+ struct etun_info *tx_info = tx_dev->priv;
+ struct net_device *rx_dev = tx_info->rx_dev;
+ if (rx_dev->flags & IFF_UP) {
+  netif_carrier_on(tx_dev);
```

```
+  netif_carrier_on(rx_dev);
+ }
+ netif_start_queue(tx_dev);
+ return 0;
+}
+
+static int etun_stop(struct net_device *tx_dev)
+{
+ struct etun_info *tx_info = tx_dev->priv;
+ struct net_device *rx_dev = tx_info->rx_dev;
+ netif_stop_queue(tx_dev);
+ if (netif_carrier_ok(tx_dev)) {
+  netif_carrier_off(tx_dev);
+  netif_carrier_off(rx_dev);
+ }
+ return 0;
+}
+
+static void etun_set_multicast_list(struct net_device *dev)
+{
+ /* Nothing sane I can do here */
+ return;
+}
+
+static int etun_ioctl(struct net_device *dev, struct ifreq *rq, int cmd)
+{
+ return -EOPNOTSUPP;
+}
+
+/* Only allow letters and numbers in an etun device name */
+static int is_valid_name(const char *name)
+{
+ const char *ptr;
+ for (ptr = name; *ptr; ptr++) {
+  if (!isalnum(*ptr))
+   return 0;
+ }
+ return 1;
+}
+
+static struct net_device *etun_alloc(net_t net, const char *name)
+{
+ struct net_device *dev;
+ struct etun_info *info;
+ int err;
+
+ if (!name || !is_valid_name(name))
+  return ERR_PTR(-EINVAL);
```

```
+
+ dev = alloc_netdev(sizeof(struct etun_info), name, ether_setup);
+ if (!dev)
+  return ERR_PTR(-ENOMEM);
+
+ info = dev->priv;
+ info->dev = dev;
+ dev->nd_net = net;
+
+ random_ether_addr(dev->dev_addr);
+ dev->tx_queue_len = 0; /* A queue is silly for a loopback device */
+ dev->hard_start_xmit = etun_xmit;
+ dev->get_stats  = etun_get_stats;
+ dev->open  = etun_open;
+ dev->stop  = etun_stop;
+ dev->set_multicast_list = etun_set_multicast_list;
+ dev->do_ioctl  = etun_ioctl;
+ dev->features  = NETIF_F_FRAGLIST
+      | NETIF_F_HIGHDMA
+      | NETIF_F_LLTX;
+ dev->flags  = IFF_BROADCAST | IFF_MULTICAST |IFF_PROMISC;
+ dev->ethtool_ops = &etun_ethtool_ops;
+ dev->destructor  = free_netdev;
+ err = register_netdev(dev);
+ if (err) {
+  free_netdev(dev);
+  dev = ERR_PTR(err);
+  goto out;
+ }
+ netif_carrier_off(dev);
+out:
+ return dev;
+}
+
+static int etun_alloc_pair(net_t net, const char *name0, const char *name1)
+{
+ struct net_device *dev0, *dev1;
+ struct etun_info *info0, *info1;
+
+ dev0 = etun_alloc(net, name0);
+ if (IS_ERR(dev0)) {
+  return PTR_ERR(dev0);
+ }
+ info0 = dev0->priv;
+
+ dev1 = etun_alloc(net, name1);
+ if (IS_ERR(dev1)) {
+  unregister_netdev(dev0);
```

```
+  return PTR_ERR(dev1);
+ }
+ info1 = dev1->priv;
+
+ dev_hold(dev0);
+ dev_hold(dev1);
+ info0->rx_dev = dev1;
+ info1->rx_dev = dev0;
+
+ /* Only place one member of the pair on the list
+  * so I don't confuse list_for_each_entry_safe,
+  * by deleting two list entries at once.
+  */
+ rtnl_lock();
+ list_add(&info0->list, &etun_list);
+ INIT_LIST_HEAD(&info1->list);
+ rtnl_unlock();
+
+ return 0;
+}
+
+static int etun_unregister_pair(struct net_device *dev0)
+{
+ struct etun_info *info0, *info1;
+ struct net_device *dev1;
+
+ ASSERT_RTNL();
+
+ if (!dev0)
+  return -ENODEV;
+
+ info0 = dev0->priv;
+ dev1  = info0->rx_dev;
+ info1 = dev1->priv;
+
+ /* Drop the cross device references */
+ dev_put(dev0);
+ dev_put(dev1);
+
+ /* Remove from the etun list */
+ if (!list_empty(&info0->list))
+  list_del_init(&info0->list);
+ if (!list_empty(&info1->list))
+  list_del_init(&info1->list);
+
+ unregister_netdevice(dev0);
+ unregister_netdevice(dev1);
+ return 0;
```

```
+}
+
+static int etun_noget(char *buffer, struct kernel_param *kp)
+{
+ return 0;
+}
+
+static int etun_newif(const char *val, struct kernel_param *kp)
+{
+ char name0[IFNAMSIZ], name1[IFNAMSIZ];
+ const char *mid;
+ int len, len0, len1;
+ if (!capable(CAP_NET_ADMIN))
+  return -EPERM;
+
+ /* Avoid frustration by removing trailing whitespace */
+ len = strlen(val);
+ while (isspace(val[len - 1]))
+  len--;
+
+ /* Split the string into 2 names */
+ mid = memchr(val, ',', len);
+ if (!mid)
+  return -EINVAL;
+
+ /* Get the first device name */
+ len0 = mid - val;
+ if (len0 > sizeof(name0) - 1)
+  len = sizeof(name0) - 1;
+ strncpy(name0, val, len0);
+ name0[len0] = '\0';
+
+ /* And the second device name */
+ len1 = len - (len0 + 1);
+ if (len1 > sizeof(name1) - 1)
+  len1 = sizeof(name1) - 1;
+ strncpy(name1, mid + 1, len1);
+ name1[len1] = '\0';
+
+ return etun_alloc_pair(current->nsproxy->net_ns, name0, name1);
+}
+
+static int etun_delif(const char *val, struct kernel_param *kp)
+{
+ char name[IFNAMSIZ];
+ int len;
+ struct net_device *dev;
+ int err;
```

```
+ if (!capable(CAP_NET_ADMIN))
+  return -EPERM;
+
+ /* Avoid frustration by removing trailing whitespace */
+ len = strlen(val);
+ while (isspace(val[len - 1]))
+  len--;
+
+ /* Get the device name */
+ if (len > sizeof(name) - 1)
+  return -EINVAL;
+ strncpy(name, val, len);
+ name[len] = '\0';
+
+ /* Double check I don't have strange characters in my device name */
+ if (!is_valid_name(name))
+  return -EINVAL;
+
+ rtnl_lock();
+ err = -ENODEV;
+ dev = __dev_get_by_name(current->nsproxy->net_ns, name);
+ err = etun_unregister_pair(dev);
+ rtnl_unlock();
+ return err;
+}
+
+static int __init etun_init(void)
+{
+ printk(KERN_INFO "etun: %s, %s\n", DRV_DESCRIPTION, DRV_VERSION);
+ printk(KERN_INFO "etun: %s\n", DRV_COPYRIGHT);
+
+ return 0;
+}
+
+static void etun_cleanup(void)
+{
+ struct etun_info *info, *tmp;
+ rtnl_lock();
+ list_for_each_entry_safe(info, tmp, &etun_list, list) {
+  etun_unregister_pair(info->dev);
+ }
+ rtnl_unlock();
+}
+
+module_param_call(newif, etun_newif, etun_noget, NULL, S_IWUSR);
+module_param_call(delif, etun_delif, etun_noget, NULL, S_IWUSR);
+module_init(etun_init);
+module_exit(etun_cleanup);
```

+MODULE_DESCRIPTION(DRV_DESCRIPTION);
+MODULE_AUTHOR("Eric Biederman <ebiederm@xmission.com>");
+MODULE_LICENSE("GPL");
--
1.4.4.1.g278f

_____
Containers mailing list
Containers@lists.osdl.org
https://lists.osdl.org/mailman/listinfo/containers