

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

## [PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

---

*Source:* <http://linux.derkeiler.com/Mailing-Lists/Kernel/2007-03/msg04470.html>

---

- *From:* "Robert P. J. Day" <rpjday@xxxxxxxxxxxxxxxx>
  - *Date:* Sat, 10 Mar 2007 03:49:52 -0500 (EST)
- 

Delete the apparently superfluous source file  
net/wanrouter/af\_wanpipe.c.

Signed-off-by: Robert P. J. Day <rpjday@xxxxxxxxxxxxxxxx>

```
diff --git a/net/wanrouter/af_wanpipe.c b/net/wanrouter/af_wanpipe.c
deleted file mode 100644
index 41d7e32..0000000
--- a/net/wanrouter/af_wanpipe.c
+++ /dev/null
@@ -1,2600 +0,0 @@
_/******
-* af_wanpipe.c WANPIPE(tm) Secure Socket Layer.
-*
-* Author: Nenad Corbic <ncorbic@xxxxxxxxxxxx>
-*
-* Copyright: (c) 2000 Sangoma Technologies Inc.
-*
-* This program is free software; you can redistribute it and/or
-* modify it under the terms of the GNU General Public License
-* as published by the Free Software Foundation; either version
-* 2 of the License, or (at your option) any later version.
-* =====
-* Due Credit:
-* Wanpipe socket layer is based on Packet and
-* the X25 socket layers. The above sockets were
-* used for the specific use of Sangoma Technologies
-* API programs.
-* Packet socket Authors: Ross Biro, Fred N. van Kempen and
-* Alan Cox.
-* X25 socket Author: Jonathan Naylor.
-* =====
-* Mar 15, 2002 Arnaldo C. Melo o Use wp_sk()->num, as it isnt anymore in sock
-* Apr 25, 2000 Nenad Corbic o Added the ability to send zero length packets.
-* Mar 13, 2000 Nenad Corbic o Added a tx buffer check via ioctl call.
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-* Mar 06, 2000 Nenad Corbic o Fixed the corrupt sock lcn problem.
-* Server and client application can run
-* simultaneously without conflicts.
-* Feb 29, 2000 Nenad Corbic o Added support for PVC protocols, such as
-* CHDLC, Frame Relay and HDLC API.
-* Jan 17, 2000 Nenad Corbic o Initial version, based on AF_PACKET socket.
-* X25API support only.
-*
-*****/
-
-#include <linux/types.h>
-#include <linux/sched.h>
-#include <linux/mm.h>
-#include <linux/capability.h>
-#include <linux/fcntl.h>
-#include <linux/socket.h>
-#include <linux/in.h>
-#include <linux/inet.h>
-#include <linux/netdevice.h>
-#include <linux/poll.h>
-#include <linux/wireless.h>
-#include <linux/kmod.h>
-#include <net/ip.h>
-#include <net/protocol.h>
-#include <linux/skbuff.h>
-#include <net/sock.h>
-#include <linux/errno.h>
-#include <linux/timer.h>
-#include <asm/system.h>
-#include <asm/uaccess.h>
-#include <linux/module.h>
-#include <linux/init.h>
-#include <linux/if_wanpipe.h>
-#include <linux/pkt_sched.h>
-#include <linux/tcp_states.h>
-#include <linux/if_wanpipe_common.h>
-
-#ifdef CONFIG_INET
-#include <net/inet_common.h>
-#endif
-
-#define SLOW_BACKOFF 0.1*HZ
-#define FAST_BACKOFF 0.01*HZ
-
-//#define PRINT_DEBUG
-#ifdef PRINT_DEBUG
-#define DBG_PRINTK(format, a...) printk(format, ## a)
-#else
-#define DBG_PRINTK(format, a...)
-#endif
-
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-
-/* SECURE SOCKET IMPLEMENTATION
- *
- * TRANSMIT:
- *
- * When the user sends a packet via send() system call
- * the wanpipe_sendmsg() function is executed.
- *
- * Each packet is enqueued into sk->sk_write_queue transmit
- * queue. When the packet is enqueued, a delayed transmit
- * timer is triggered which acts as a Bottom Half handler.
- *
- * wanpipe_delay_transmit() function (BH), dequeues packets
- * from the sk->sk_write_queue transmit queue and sends it
- * to the driver via dev->hard_start_xmit(skb, dev) function.
- * Note, this function is actual a function pointer of if_send()
- * routine in the wanpipe driver.
- *
- * X25API GUARANTEED DELIVERY:
- *
- * In order to provide 100% guaranteed packet delivery,
- * an atomic 'packet_sent' counter is implemented. Counter
- * is incremented for each packet enqueued
- * into sk->sk_write_queue. Counter is decremented each
- * time wanpipe_delayed_transmit() function successfully
- * passes the packet to the driver. Before each send(), a poll
- * routine checks the sock resources. The maximum value of
- * packet sent counter is 1, thus if one packet is queued, the
- * application will block until that packet is passed to the
- * driver.
- *
- * RECEIVE:
- *
- * Wanpipe device drivers call the socket bottom half
- * function, wanpipe_rcv() to queue the incoming packets
- * into an AF_WANPIPE socket queue. Based on wanpipe_rcv()
- * return code, the driver knows whether the packet was
- * successfully queued. If the socket queue is full,
- * protocol flow control is used by the driver, if any,
- * to slow down the traffic until the sock queue is free.
- *
- * Every time a packet arrives into a socket queue the
- * socket wakes up processes which are waiting to receive
- * data.
- *
- * If the socket queue is full, the driver sets a block
- * bit which signals the socket to kick the wanpipe driver
- * bottom half handler when the socket queue is partially
- * empty. wanpipe_recvmsg() function performs this action.
- *
- * In case of x25api, packets will never be dropped, since
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- * flow control is available.
- *
- * In case of streaming protocols like CHDLC, packets will
- * be dropped but the statistics will be generated.
- */
-
-
-/* The code below is used to test memory leaks. It prints out
- * a message every time kcalloc and kfree system calls get executed.
- * If the calls match there is no leak :)
- */
-
-/******FOR DEBUGGING PURPOSES*****
-#define KMEM_SAFETYZONE 8
-
-static void * dbg_kcalloc(unsigned int size, int prio, int line) {
- void * v = kcalloc(size,prio);
- printk(KERN_INFO "line %d kcalloc(%d,%d) = %p\n",line,size,prio,v);
- return v;
-}
-static void dbg_kfree(void * v, int line) {
- printk(KERN_INFO "line %d kfree(%p)\n",line,v);
- kfree(v);
-}
-
-#define kcalloc(x,y) dbg_kcalloc(x,y,__LINE__)
-#define kfree(x) dbg_kfree(x,__LINE__)
-*****/
-
-
-/* List of all wanpipe sockets. */
-#LIST_HEAD(wanpipe_sklist);
-#static DEFINE_RWLOCK(wanpipe_sklist_lock);
-
-#atomic_t wanpipe_socks_nr;
-#static unsigned long wanpipe_tx_critical;
-
-#if 0
-/* Private wanpipe socket structures. */
-#struct wanpipe_opt
-#{
- void *mbox; /* Mail box */
- void *card; /* Card bouded to */
- struct net_device *dev; /* Bounded device */
- unsigned short lcn; /* Binded LCN */
- unsigned char svc; /* 0=pvc, 1=svc */
- unsigned char timer; /* flag for delayed transmit*/
- struct timer_list tx_timer;
- unsigned poll_cnt;
- unsigned char force; /* Used to force sock release */
- atomic_t packet_sent;
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-};
-#endif
-
-static int sk_count;
-extern const struct proto_ops wanpipe_ops;
-static unsigned long find_free_critical;
-
-static void wanpipe_unlink_driver(struct sock *sk);
-static void wanpipe_link_driver(struct net_device *dev, struct sock *sk);
-static void wanpipe_wakeup_driver(struct sock *sk);
-static int execute_command(struct sock *, unsigned char, unsigned int);
-static int check_dev(struct net_device *dev, sdla_t *card);
-struct net_device *wanpipe_find_free_dev(sdla_t *card);
-static void wanpipe_unlink_card (struct sock *);
-static int wanpipe_link_card (struct sock *);
-static struct sock *wanpipe_make_new(struct sock *);
-static struct sock *wanpipe_alloc_socket(void);
-static inline int get_atomic_device(struct net_device *dev);
-static int wanpipe_exec_cmd(struct sock *, int, unsigned int);
-static int get_ioctl_cmd (struct sock *, void *);
-static int set_ioctl_cmd (struct sock *, void *);
-static void release_device(struct net_device *dev);
-static void wanpipe_kill_sock_timer (unsigned long data);
-static void wanpipe_kill_sock_irq (struct sock *);
-static void wanpipe_kill_sock_accept (struct sock *);
-static int wanpipe_do_bind(struct sock *sk, struct net_device *dev,
- int protocol);
-struct sock * get_newsk_from_skb (struct sk_buff *);
-static int wanpipe_debug (struct sock *, void *);
-static void wanpipe_delayed_transmit (unsigned long data);
-static void release_driver(struct sock *);
-static void start_cleanup_timer (struct sock *);
-static void check_write_queue(struct sock *);
-static int check_driver_busy (struct sock *);
-
-/*=====
- * wanpipe_rcv
- *
- * Wanpipe socket bottom half handler. This function
- * is called by the WANPIPE device drivers to queue a
- * incoming packet into the socket receive queue.
- * Once the packet is queued, all processes waiting to
- * read are woken up.
- *
- * During socket bind, this function is bounded into
- * WANPIPE driver private.
- *=====*/
-
-static int wanpipe_rcv(struct sk_buff *skb, struct net_device *dev,
- struct sock *sk)
-{
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- struct wan_sockaddr_ll *sll = (struct wan_sockaddr_ll*)skb->cb;
- wanpipe_common_t *chan = dev->priv;
- /*
- * When we registered the protocol we saved the socket in the data
- * field for just this event.
- */
-
- skb->dev = dev;
-
- sll->sll_family = AF_WANPIPE;
- sll->sll_hatype = dev->type;
- sll->sll_protocol = skb->protocol;
- sll->sll_pkttype = skb->pkt_type;
- sll->sll_ifindex = dev->ifindex;
- sll->sll_halen = 0;
-
- if (dev->hard_header_parse)
- sll->sll_halen = dev->hard_header_parse(skb, sll->sll_addr);
-
- /*
- * WAN_PACKET_DATA : Data which should be passed up the receive queue.
- * WAN_PACKET_ASYNC : Asynchronous data like place call, which should
- * be passed up the listening sock.
- * WAN_PACKET_ERR : Asynchronous data like clear call or restart
- * which should go into an error queue.
- */
- switch (skb->pkt_type){
-
- case WAN_PACKET_DATA:
- if (sock_queue_rcv_skb(sk,skb)<0){
- return -ENOMEM;
- }
- break;
- case WAN_PACKET_CMD:
- sk->sk_state = chan->state;
- /* Bug fix: update Mar6.
- * Do not set the sock lcn number here, since
- * cmd is not guaranteed to be executed on the
- * board, thus Lcn could be wrong */
- sk->sk_data_ready(sk, skb->len);
- kfree_skb(skb);
- break;
- case WAN_PACKET_ERR:
- sk->sk_state = chan->state;
- if (sock_queue_err_skb(sk,skb)<0){
- return -ENOMEM;
- }
- break;
- default:
- printk(KERN_INFO "wansock: BH Illegal Packet Type Dropping\n");
- kfree_skb(skb);
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- break;
- }
-
-//????????????????????
-// if (sk->sk_state == WANSOCK_DISCONNECTED){
-// if (sk->sk_zapped) {
-// //printk(KERN_INFO "wansock: Disconnected, killing early\n");
-// wanpipe_unlink_driver(sk);
-// sk->sk_bound_dev_if = 0;
-// }
-// }
-
- return 0;
-}
-
-/*=====
- * wanpipe_listen_rcv
- *
- * Wanpipe LISTEN socket bottom half handler. This function
- * is called by the WANPIPE device drivers to queue an
- * incoming call into the socket listening queue.
- * Once the packet is queued, the waiting accept() process
- * is woken up.
- *
- * During socket bind, this function is bounded into
- * WANPIPE driver private.
- *
- * IMPORTANT NOTE:
- * The accept call() is waiting for an skb packet
- * which contains a pointer to a device structure.
- *
- * When we do a bind to a device structre, we
- * bind a newly created socket into "chan->sk". Thus,
- * when accept receives the skb packet, it will know
- * from which dev it came form, and in turn it will know
- * the address of the new sock.
- *
- * NOTE: This function gets called from driver ISR.
- *=====*/
-
-static int wanpipe_listen_rcv (struct sk_buff *skb, struct sock *sk)
-{
- wanpipe_opt *wp = wp_sk(sk), *newwp;
- struct wan_sockaddr_ll *sll = (struct wan_sockaddr_ll*)skb->cb;
- struct sock *newsk;
- struct net_device *dev;
- sdla_t *card;
- mbox_cmd_t *mbox_ptr;
- wanpipe_common_t *chan;
-
- /* Find a free device, if none found, all svc's are busy
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- */
-
- card = (sdla_t*)wp->card;
- if (!card){
- printk(KERN_INFO "wansock: LISTEN ERROR, No Card\n");
- return -ENODEV;
- }
-
- dev = wanpipe_find_free_dev(card);
- if (!dev){
- printk(KERN_INFO "wansock: LISTEN ERROR, No Free Device\n");
- return -ENODEV;
- }
-
- chan=dev->priv;
- chan->state = WANSOCK_CONNECTING;
-
- /* Allocate a new sock, which accept will bind
- * and pass up to the user
- */
- if ((newsk = wanpipe_make_new(sk)) == NULL){
- release_device(dev);
- return -ENOMEM;
- }
-
-
- /* Initialize the new sock structure
- */
- newsk->sk_bound_dev_if = dev->ifindex;
- newwp = wp_sk(newsk);
- newwp->card = wp->card;
-
- /* Insert the sock into the main wanpipe
- * sock list.
- */
- atomic_inc(&wanpipe_socks_nr);
-
- /* Allocate and fill in the new Mail Box. Then
- * bind the mail box to the sock. It will be
- * used by the ioctl call to read call information
- * and to execute commands.
- */
- if ((mbox_ptr = kzalloc(sizeof(mbox_cmd_t), GFP_ATOMIC)) == NULL) {
- wanpipe_kill_sock_irq (newsk);
- release_device(dev);
- return -ENOMEM;
- }
- memcpy(mbox_ptr,skb->data,skb->len);
-
- /* Register the lcn on which incoming call came
- * from. Thus, if we have to clear it, we know
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- * which lcn to clear
- */
-
- newwp->lcn = mbox_ptr->cmd.lcn;
- newwp->mbox = (void *)mbox_ptr;
-
- DBG_PRINTK(KERN_INFO "NEWSOCK : Device %s, bind to lcn %i\n",
- dev->name, mbox_ptr->cmd.lcn);
-
- chan->lcn = mbox_ptr->cmd.lcn;
- card->u.x.svc_to_dev_map[(chan->lcn%MAX_X25_LCN)] = dev;
-
- sock_reset_flag(newsk, SOCK_ZAPPED);
- newwp->num = htons(X25_PROT);
-
- if (wanpipe_do_bind(newsk, dev, newwp->num)) {
- wanpipe_kill_sock_irq (newsk);
- release_device(dev);
- return -EINVAL;
- }
- newsk->sk_state = WANSOCK_CONNECTING;
-
-
- /* Fill in the standard sock address info */
-
- sll->sll_family = AF_WANPIPE;
- sll->sll_hatype = dev->type;
- sll->sll_protocol = skb->protocol;
- sll->sll_pkttype = skb->pkt_type;
- sll->sll_ifindex = dev->ifindex;
- sll->sll_halen = 0;
-
-
- skb->dev = dev;
- sk->sk_ack_backlog++;
-
- /* We must do this manually, since the sock_queue_rcv_skb()
- * function sets the skb->dev to NULL. However, we use
- * the dev field in the accept function.*/
- if (atomic_read(&sk->sk_rmem_alloc) + skb->truesize >=
- (unsigned)sk->sk_rcvbuf) {
-
- wanpipe_unlink_driver(newsk);
- wanpipe_kill_sock_irq (newsk);
- --sk->sk_ack_backlog;
- return -ENOMEM;
- }
-
-
- skb_set_owner_r(skb, sk);
- skb_queue_tail(&sk->sk_receive_queue, skb);
- sk->sk_data_ready(sk, skb->len);
-
-
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- return 0;
- }
-
-
-
-/*=====
- * wanpipe_make_new
- *
- * Create a new sock, and allocate a wanpipe private
- * structure to it. Also, copy the important data
- * from the original sock to the new sock.
- *
- * This function is used by wanpipe_listen_rcv() listen
- * bottom half handler. A copy of the listening sock
- * is created using this function.
- *
- *=====*/
-
-static struct sock *wanpipe_make_new(struct sock *osk)
- {
- struct sock *sk;
-
- if (osk->sk_type != SOCK_RAW)
- return NULL;
-
- if ((sk = wanpipe_alloc_socket()) == NULL)
- return NULL;
-
- sk->sk_type = osk->sk_type;
- sk->sk_socket = osk->sk_socket;
- sk->sk_priority = osk->sk_priority;
- sk->sk_protocol = osk->sk_protocol;
- wp_sk(sk)->num = wp_sk(osk)->num;
- sk->sk_rcvbuf = osk->sk_rcvbuf;
- sk->sk_sndbuf = osk->sk_sndbuf;
- sk->sk_state = WANSOCK_CONNECTING;
- sk->sk_sleep = osk->sk_sleep;
-
- if (sock_flag(osk, SOCK_DBG))
- sock_set_flag(sk, SOCK_DBG);
-
- return sk;
- }
-
-/*
- * FIXME: wanpipe_opt has to include a sock in its definition and stop using
- * sk_protinfo, but this code is not even compilable now, so lets leave it for
- * later.
- */
-static struct proto wanpipe_proto = {
- .name = "WANPIPE",
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- .owner = THIS_MODULE,
- .obj_size = sizeof(struct sock),
-};
-
-/*=====
- * wanpipe_make_new
- *
- * Allocate memory for the a new sock, and sock
- * private data.
- *
- * Increment the module use count.
- *
- * This function is used by wanpipe_create() and
- * wanpipe_make_new() functions.
- *
- *=====*/
-
-static struct sock *wanpipe_alloc_socket(void)
-{
- struct sock *sk;
- struct wanpipe_opt *wan_opt;
-
- if ((sk = sk_alloc(PF_WANPIPE, GFP_ATOMIC, &wanpipe_proto, 1)) == NULL)
- return NULL;
-
- if ((wan_opt = kzalloc(sizeof(struct wanpipe_opt), GFP_ATOMIC)) == NULL) {
- sk_free(sk);
- return NULL;
- }
-
- wp_sk(sk) = wan_opt;
-
- /* Use timer to send data to the driver. This will act
- * as a BH handler for sendmsg functions */
- init_timer(&wan_opt->tx_timer);
- wan_opt->tx_timer.data = (unsigned long)sk;
- wan_opt->tx_timer.function = wanpipe_delayed_transmit;
-
- sock_init_data(NULL, sk);
- return sk;
-}
-
-/*=====
- * wanpipe_sendmsg
- *
- * This function implements a sendto() system call,
- * for AF_WANPIPE socket family.
- * During socket bind() sk->sk_bound_dev_if is initialized
- * to a correct network device. This number is used
- * to find a network device to which the packet should
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- * be passed to.
- *
- * Each packet is queued into sk->sk_write_queue and
- * delayed transmit bottom half handler is marked for
- * execution.
- *
- * A socket must be in WANSOCK_CONNECTED state before
- * a packet is queued into sk->sk_write_queue.
- *=====*/
-
-static int wanpipe_sendmsg(struct kiocb *iocb, struct socket *sock,
- struct msghdr *msg, int len)
- {
-     wanpipe_opt *wp;
-     struct sock *sk = sock->sk;
-     struct wan_sockaddr_ll *saddr=(struct wan_sockaddr_ll *)msg->msg_name;
-     struct sk_buff *skb;
-     struct net_device *dev;
-     unsigned short proto;
-     unsigned char *addr;
-     int ifindex, err, reserve = 0;
-
-     if (!sock_flag(sk, SOCK_ZAPPED))
-         return -ENETDOWN;
-
-     if (sk->sk_state != WANSOCK_CONNECTED)
-         return -ENOTCONN;
-
-     if (msg->msg_flags & ~(MSG_DONTWAIT|MSG_CMSG_COMPAT))
-         return(-EINVAL);
-
-     /* it was <=, now one can send
-     * zero length packets */
-     if (len < sizeof(x25api_hdr_t))
-         return -EINVAL;
-
-     wp = wp_sk(sk);
-
-     if (saddr == NULL) {
-         ifindex = sk->sk_bound_dev_if;
-         proto = wp->num;
-         addr = NULL;
-
-     } else {
-         if (msg->msg_namelen < sizeof(struct wan_sockaddr_ll)){
-             return -EINVAL;
-         }
-
-         ifindex = sk->sk_bound_dev_if;
-         proto = saddr->sll_protocol;
-     }
- }
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- addr = saddr->sl_addr;
- }
-
- dev = dev_get_by_index(ifindex);
- if (dev == NULL){
- printk(KERN_INFO "wansock: Send failed, dev index: %i\n",ifindex);
- return -ENXIO;
- }
- dev_put(dev);
-
- if (sock->type == SOCK_RAW)
- reserve = dev->hard_header_len;
-
- if (len > dev->mtu+reserve){
- return -EMSGSIZE;
- }
-
- skb = sock_alloc_send_skb(sk, len + LL_RESERVED_SPACE(dev),
- msg->msg_flags & MSG_DONTWAIT, &err);
-
- if (skb==NULL){
- goto out_unlock;
- }
-
- skb_reserve(skb, LL_RESERVED_SPACE(dev));
- skb->nh.raw = skb->data;
-
- /* Returns -EFAULT on error */
- err = memcpy_fromiovec(skb_put(skb,len), msg->msg_iov, len);
- if (err){
- goto out_free;
- }
-
- if (dev->hard_header) {
- int res;
- err = -EINVAL;
- res = dev->hard_header(skb, dev, ntohs(proto), addr, NULL, len);
- if (res<0){
- goto out_free;
- }
- }
-
- skb->protocol = proto;
- skb->dev = dev;
- skb->priority = sk->sk_priority;
- skb->pkt_type = WAN_PACKET_DATA;
-
- err = -ENETDOWN;
- if (!(dev->flags & IFF_UP))
- goto out_free;
-
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- if (atomic_read(&sk->sk_wmem_alloc) + skb->truesize >
- (unsigned int)sk->sk_sndbuf){
- kfree_skb(skb);
- return -ENOBUFS;
- }
-
- skb_queue_tail(&sk->sk_write_queue,skb);
- atomic_inc(&wp->packet_sent);
-
- if (!(test_and_set_bit(0, &wp->timer)))
- mod_timer(&wp->tx_timer, jiffies + 1);
-
- return(len);
-
-out_free:
- kfree_skb(skb);
-out_unlock:
- return err;
-}
-
-/*=====
- * wanpipe_delayed_transmit
- *
- * Transmit bottom half handler. It dequeues packets
- * from sk->sk_write_queue and passes them to the
- * driver. If the driver is busy, the packet is
- * re-enqueued.
- *
- * Packet Sent counter is decremented on successful
- * transmission.
- *=====*/
-
-
-static void wanpipe_delayed_transmit (unsigned long data)
-{
- struct sock *sk=(struct sock *)data;
- struct sk_buff *skb;
- wanpipe_opt *wp = wp_sk(sk);
- struct net_device *dev = wp->dev;
- sdla_t *card = (sdla_t*)wp->card;
-
- if (!card || !dev){
- clear_bit(0, &wp->timer);
- DBG_PRINTK(KERN_INFO "wansock: Transmit delay, no dev or card\n");
- return;
- }
-
- if (sk->sk_state != WANSOCK_CONNECTED || !sock_flag(sk, SOCK_ZAPPED)) {
- clear_bit(0, &wp->timer);
- DBG_PRINTK(KERN_INFO "wansock: Tx Timer, State not CONNECTED\n");
- return;
- }
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- }
-
- /* If driver is executing command, we must offload
- * the board by not sending data. Otherwise a
- * pending command will never get a free buffer
- * to execute */
- if (atomic_read(&card->u.x.command_busy)){
- wp->tx_timer.expires = jiffies + SLOW_BACKOFF;
- add_timer(&wp->tx_timer);
- DBG_PRINTK(KERN_INFO "wansock: Tx Timer, command bys BACKOFF\n");
- return;
- }
-
-
- if (test_and_set_bit(0,&wanpipe_tx_critical)){
- printk(KERN_INFO "WanSock: Tx timer critical %s\n",dev->name);
- wp->tx_timer.expires = jiffies + SLOW_BACKOFF;
- add_timer(&wp->tx_timer);
- return;
- }
-
- /* Check for a packet in the fifo and send */
- if ((skb = skb_dequeue(&sk->sk_write_queue)) != NULL){
-
- if (dev->hard_start_xmit(skb, dev) != 0){
-
- /* Driver failed to transmit, re-enqueue
- * the packet and retry again later */
- skb_queue_head(&sk->sk_write_queue,skb);
- clear_bit(0,&wanpipe_tx_critical);
- return;
- }else{
-
- /* Packet Sent successful. Check for more packets
- * if more packets, re-trigger the transmit routine
- * other wise exit
- */
- atomic_dec(&wp->packet_sent);
-
- if (skb_peek(&sk->sk_write_queue) == NULL) {
- /* If there is nothing to send, kick
- * the poll routine, which will trigger
- * the application to send more data */
- sk->sk_data_ready(sk, 0);
- clear_bit(0, &wp->timer);
- }else{
- /* Reschedule as fast as possible */
- wp->tx_timer.expires = jiffies + 1;
- add_timer(&wp->tx_timer);
- }
- }
- }
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- }
- clear_bit(0,&wanpipe_tx_critical);
- }
-
- /*=====
- * execute_command
- *
- * Execute x25api commands. The atomic variable
- * chan->command is used to indicate to the driver that
- * command is pending for execution. The actual command
- * structure is placed into a sock mbox structure
- * (wp_sk(sk)->mbox).
- *
- * The sock private structure, mbox is
- * used as shared memory between sock and the driver.
- * Driver uses the sock mbox to execute the command
- * and return the result.
- *
- * For all command except PLACE CALL, the function
- * waits for the result. PLACE CALL can be either
- * blocking or nonblocking. The user sets this option
- * via ioctl call.
- *=====*/
-
-
- static int execute_command(struct sock *sk, unsigned char cmd, unsigned int flags)
- {
-     wanpipe_opt *wp = wp_sk(sk);
-     struct net_device *dev;
-     wanpipe_common_t *chan=NULL;
-     int err=0;
-     DECLARE_WAITQUEUE(wait, current);
-
-     dev = dev_get_by_index(sk->sk_bound_dev_if);
-     if (dev == NULL){
-         printk(KERN_INFO "wansock: Exec failed no dev %i\n",
-             sk->sk_bound_dev_if);
-         return -ENODEV;
-     }
-     dev_put(dev);
-
-     if ((chan=dev->priv) == NULL){
-         printk(KERN_INFO "wansock: Exec cmd failed no priv area\n");
-         return -ENODEV;
-     }
-
-     if (atomic_read(&chan->command)){
-         printk(KERN_INFO "wansock: ERROR: Command already running %x, %s\n",
-             atomic_read(&chan->command),dev->name);
-         return -EINVAL;
-     }
- }
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-
- if (!wp->mbox) {
- printk(KERN_INFO "wansock: In execute without MBOX\n");
- return -EINVAL;
- }
-
- ((mbox_cmd_t*)wp->mbox)->cmd.command = cmd;
- ((mbox_cmd_t*)wp->mbox)->cmd.lcn = wp->lc;
- ((mbox_cmd_t*)wp->mbox)->cmd.result = 0x7F;
-
-
- if (flags & O_NONBLOCK){
- cmd |= 0x80;
- atomic_set(&chan->command, cmd);
- }else{
- atomic_set(&chan->command, cmd);
- }
-
- add_wait_queue(sk->sk_sleep,&wait);
- current->state = TASK_INTERRUPTIBLE;
- for (;;) {
- if (((mbox_cmd_t*)wp->mbox)->cmd.result != 0x7F) {
- err = 0;
- break;
- }
- if (signal_pending(current)) {
- err = -ERESTARTSYS;
- break;
- }
- schedule();
- }
- current->state = TASK_RUNNING;
- remove_wait_queue(sk->sk_sleep,&wait);
-
- return err;
- }
-
- /*=====
- * wanpipe_destroy_timer
- *
- * Used by wanpipe_release, to delay release of
- * the socket.
- *=====*/
-
- static void wanpipe_destroy_timer(unsigned long data)
- {
- struct sock *sk=(struct sock *)data;
- wanpipe_opt *wp = wp_sk(sk);
-
- if ((!atomic_read(&sk->sk_wmem_alloc) &&
- !atomic_read(&sk->sk_rmem_alloc)) ||
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- (++wp->force == 5) {
-
- if (atomic_read(&sk->sk_wmem_alloc) ||
- atomic_read(&sk->sk_rmem_alloc))
- printk(KERN_INFO "wansock: Warning, Packet Discarded due to sock shutdown!\n");
-
- kfree(wp);
- wp_sk(sk) = NULL;
-
- if (atomic_read(&sk->sk_refcnt) != 1) {
- atomic_set(&sk->sk_refcnt, 1);
- DBG_PRINTK(KERN_INFO "wansock: Error, wrong reference count: %i ! :delay.\n",
- atomic_read(&sk->sk_refcnt));
- }
- sock_put(sk);
- atomic_dec(&wanpipe_socks_nr);
- return;
- }
-
- sk->sk_timer.expires = jiffies + 5 * HZ;
- add_timer(&sk->sk_timer);
- printk(KERN_INFO "wansock: packet sk destroy delayed\n");
- }
-
- /*=====
- * wanpipe_unlink_driver
- *
- * When the socket is released, this function is
- * used to remove links that bind the sock and the
- * driver together.
- *=====*/
-static void wanpipe_unlink_driver (struct sock *sk)
- {
- struct net_device *dev;
- wanpipe_common_t *chan=NULL;
-
- sock_reset_flag(sk, SOCK_ZAPPED);
- sk->sk_state = WANSOCK_DISCONNECTED;
- wp_sk(sk)->dev = NULL;
-
- dev = dev_get_by_index(sk->sk_bound_dev_if);
- if (!dev){
- printk(KERN_INFO "wansock: No dev on release\n");
- return;
- }
- dev_put(dev);
-
- if ((chan = dev->priv) == NULL){
- printk(KERN_INFO "wansock: No Priv Area on release\n");
- return;
- }
- }
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-
- set_bit(0,&chan->common_critical);
- chan->sk=NULL;
- chan->func=NULL;
- chan->mbox=NULL;
- chan->tx_timer=NULL;
- clear_bit(0,&chan->common_critical);
- release_device(dev);
-
- return;
-}
-
-/*=====
- * wanpipe_link_driver
- *
- * Upon successful bind(), sock is linked to a driver
- * by binding in the wanpipe_rcv() bottom half handler
- * to the driver function pointer, as well as sock and
- * sock mailbox addresses. This way driver can pass
- * data up the socket.
- *=====*/
-
-static void wanpipe_link_driver(struct net_device *dev, struct sock *sk)
-{
- wanpipe_opt *wp = wp_sk(sk);
- wanpipe_common_t *chan = dev->priv;
- if (!chan)
- return;
- set_bit(0,&chan->common_critical);
- chan->sk=sk;
- chan->func=wanpipe_rcv;
- chan->mbox = wp->mbox;
- chan->tx_timer = &wp->tx_timer;
- wp->dev = dev;
- sock_set_flag(sk, SOCK_ZAPPED);
- clear_bit(0,&chan->common_critical);
-}
-
-/*=====
- * release_device
- *
- * During sock release, clear a critical bit, which
- * marks the device a being taken.
- *=====*/
-
-static void release_device(struct net_device *dev)
-{
- wanpipe_common_t *chan=dev->priv;
- clear_bit(0,(void*)&chan->rw_bind);
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-}
-
-/*=====
- * wanpipe_release
- *
- * Close a PACKET socket. This is fairly simple. We
- * immediately go to 'closed' state and remove our
- * protocol entry in the device list.
- *=====*/
-
-static int wanpipe_release(struct socket *sock)
-{
- wanpipe_opt *wp;
- struct sock *sk = sock->sk;
-
- if (!sk)
- return 0;
-
- wp = wp_sk(sk);
- check_write_queue(sk);
-
- /* Kill the tx timer, if we don't kill it now, the timer
- * will run after we kill the sock. Timer code will
- * try to access the sock which has been killed and cause
- * kernel panic */
-
- del_timer(&wp->tx_timer);
-
- /*
- * Unhook packet receive handler.
- */
-
- if (wp->num == htons(X25_PROT) &&
- sk->sk_state != WANSOCK_DISCONNECTED && sock_flag(sk, SOCK_ZAPPED)) {
- struct net_device *dev = dev_get_by_index(sk->sk_bound_dev_if);
- wanpipe_common_t *chan;
- if (dev){
- chan=dev->priv;
- atomic_set(&chan->disconnect,1);
- DBG_PRINTK(KERN_INFO "wansock: Sending Clear Indication %i\n",
- sk->sk_state);
- dev_put(dev);
- }
- }
-
- set_bit(1,&wanpipe_tx_critical);
- write_lock(&wanpipe_sklist_lock);
- sk_del_node_init(sk);
- write_unlock(&wanpipe_sklist_lock);
- clear_bit(1,&wanpipe_tx_critical);
-
-}
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-
-
- release_driver(sk);
-
-
- /*
- * Now the socket is dead. No more input will appear.
- */
-
- sk->sk_state_change(sk); /* It is useless. Just for sanity. */
-
- sock->sk = NULL;
- sk->sk_socket = NULL;
- sock_set_flag(sk, SOCK_DEAD);
-
- /* Purge queues */
- skb_queue_purge(&sk->sk_receive_queue);
- skb_queue_purge(&sk->sk_write_queue);
- skb_queue_purge(&sk->sk_error_queue);
-
- if (atomic_read(&sk->sk_rmem_alloc) ||
-     atomic_read(&sk->sk_wmem_alloc)) {
-     del_timer(&sk->sk_timer);
-     printk(KERN_INFO "wansock: Killing in Timer R %i , W %i\n",
-            atomic_read(&sk->sk_rmem_alloc),
-            atomic_read(&sk->sk_wmem_alloc));
-     sk->sk_timer.data = (unsigned long)sk;
-     sk->sk_timer.expires = jiffies + HZ;
-     sk->sk_timer.function = wanpipe_destroy_timer;
-     add_timer(&sk->sk_timer);
-     return 0;
- }
-
- kfree(wp);
- wp_sk(sk) = NULL;
-
- if (atomic_read(&sk->sk_refcnt) != 1) {
-     DBG_PRINTK(KERN_INFO "wansock: Error, wrong reference count: %i !:release.\n",
-               atomic_read(&sk->sk_refcnt));
-     atomic_set(&sk->sk_refcnt, 1);
- }
- sock_put(sk);
- atomic_dec(&wanpipe_socks_nr);
- return 0;
-}
-
-/*=====
- * check_write_queue
- *
- * During sock shutdown, if the sock state is
- * WANSOCK_CONNECTED and there is transmit data
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- * pending. Wait until data is released
- * before proceeding.
- *=====*/
-
-static void check_write_queue(struct sock *sk)
-{
-
- if (sk->sk_state != WANSOCK_CONNECTED)
- return;
-
- if (!atomic_read(&sk->sk_wmem_alloc))
- return;
-
- printk(KERN_INFO "wansock: MAJOR ERROR, Data lost on sock release !!!\n");
-
-}
-
-/*=====*/
- * release_driver
- *
- * This function is called during sock shutdown, to
- * release any resources and links that bind the sock
- * to the driver. It also changes the state of the
- * sock to WANSOCK_DISCONNECTED
- *=====*/
-
-static void release_driver(struct sock *sk)
-{
- wanpipe_opt *wp;
- struct sk_buff *skb=NULL;
- struct sock *deadsk=NULL;
-
- if (sk->sk_state == WANSOCK_LISTEN ||
- sk->sk_state == WANSOCK_BIND_LISTEN) {
- while ((skb = skb_dequeue(&sk->sk_receive_queue)) != NULL) {
- if ((deadsk = get_newsk_from_skb(skb)){
- DBG_PRINTK (KERN_INFO "wansock: RELEASE: FOUND DEAD SOCK\n");
- sock_set_flag(deadsk, SOCK_DEAD);
- start_cleanup_timer(deadsk);
- }
- kfree_skb(skb);
- }
- if (sock_flag(sk, SOCK_ZAPPED))
- wanpipe_unlink_card(sk);
- }else{
- if (sock_flag(sk, SOCK_ZAPPED))
- wanpipe_unlink_driver(sk);
- }
- sk->sk_state = WANSOCK_DISCONNECTED;
- sk->sk_bound_dev_if = 0;
- sock_reset_flag(sk, SOCK_ZAPPED);
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".



[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- }
-
- write_lock(&wanpipe_splist_lock);
- sk_del_node_init(sk);
- write_unlock(&wanpipe_splist_lock);
-
-
- if (wp_sk(sk)->num == htons(X25_PROT) &&
- sk->sk_state != WANSOCK_DISCONNECTED) {
- struct net_device *dev = dev_get_by_index(sk->sk_bound_dev_if);
- wanpipe_common_t *chan;
- if (dev){
- chan=dev->priv;
- atomic_set(&chan->disconnect,1);
- dev_put(dev);
- }
- }
-
- release_driver(sk);
-
- sk->sk_socket = NULL;
-
- /* Purge queues */
- skb_queue_purge(&sk->sk_receive_queue);
- skb_queue_purge(&sk->sk_write_queue);
- skb_queue_purge(&sk->sk_error_queue);
-
- if (atomic_read(&sk->sk_rmem_alloc) ||
- atomic_read(&sk->sk_wmem_alloc)) {
- del_timer(&sk->sk_timer);
- printk(KERN_INFO "wansock: Killing SOCK in Timer\n");
- sk->sk_timer.data = (unsigned long)sk;
- sk->sk_timer.expires = jiffies + HZ;
- sk->sk_timer.function = wanpipe_destroy_timer;
- add_timer(&sk->sk_timer);
- return;
- }
-
- kfree(wp_sk(sk));
- wp_sk(sk) = NULL;
-
- if (atomic_read(&sk->sk_refcnt) != 1) {
- atomic_set(&sk->sk_refcnt, 1);
- DBG_PRINTK(KERN_INFO "wansock: Error, wrong reference count: %i ! :timer.\n",
- atomic_read(&sk->sk_refcnt));
- }
- sock_put(sk);
- atomic_dec(&wanpipe_socks_nr);
- return;
-}
-
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".



[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-
-
-/*=====
- * wanpipe_do_bind
- *
- * Bottom half of the binding system call.
- * Once the wanpipe_bind() function checks the
- * legality of the call, this function binds the
- * sock to the driver.
- *=====*/
-
-static int wanpipe_do_bind(struct sock *sk, struct net_device *dev,
- int protocol)
-{
- wanpipe_opt *wp = wp_sk(sk);
- wanpipe_common_t *chan=NULL;
- int err=0;
-
- if (sock_flag(sk, SOCK_ZAPPED)) {
- err = -EALREADY;
- goto bind_unlock_exit;
- }
-
- wp->num = protocol;
-
- if (protocol == 0){
- release_device(dev);
- err = -EINVAL;
- goto bind_unlock_exit;
- }
-
- if (dev) {
- if (dev->flags&IFF_UP) {
- chan=dev->priv;
- sk->sk_state = chan->state;
-
- if (wp->num == htons(X25_PROT) &&
- sk->sk_state != WANSOCK_DISCONNECTED &&
- sk->sk_state != WANSOCK_CONNECTING) {
- DBG_PRINTK(KERN_INFO
- "wansock: Binding to Device not DISCONNECTED %i\n",
- sk->sk_state);
- release_device(dev);
- err = -EAGAIN;
- goto bind_unlock_exit;
- }
-
- wanpipe_link_driver(dev,sk);
- sk->sk_bound_dev_if = dev->ifindex;
-
- /* X25 Specific option */
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".



[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- card = wanpipe_find_card (sll->sll_card);
- if (!card){
- printk(KERN_INFO "wansock: Wanpipe card not found: %s\n",sll->sll_card);
- return -ENODEV;
- }else{
- wp_sk(sk)->card = (void *)card;
- }
-
- if (!strcmp(sll->sll_device,"svc_listen")){
-
- /* Bind a sock to a card structure for listening
- */
- int err=0;
-
- /* This is x25 specific area if protocol doesn't
- * match, return error */
- if (sll->sll_protocol != htons(X25_PROT))
- return -EINVAL;
-
- err= wanpipe_link_card (sk);
- if (err < 0)
- return err;
-
- if (sll->sll_protocol)
- wp->num = sll->sll_protocol;
- sk->sk_state = WANSOCK_BIND_LISTEN;
- return 0;
-
- }else if (!strcmp(sll->sll_device,"svc_connect")){
-
- /* This is x25 specific area if protocol doesn't
- * match, return error */
- if (sll->sll_protocol != htons(X25_PROT))
- return -EINVAL;
-
- /* Find a free device
- */
- dev = wanpipe_find_free_dev(card);
- if (dev == NULL){
- DBG_PRINTK(KERN_INFO "wansock: No free network devices for card %s\n",
- card->devname);
- return -EINVAL;
- }
- }else{
- /* Bind a socket to a interface name
- * This is used by PVC mostly
- */
- strncpy(name,sll->sll_device,sizeof(name));
- dev = dev_get_by_name(name);
- if (dev == NULL){
- printk(KERN_INFO "wansock: Failed to get Dev from name: %s,\n",
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".



[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- return 1;
-}
-
-/*=====
- * wanpipe_find_free_dev
- *
- * Find a free network interface. If found set atomic
- * bit indicating that the interface is taken.
- * X25API Specific.
- *=====*/
-
-struct net_device *wanpipe_find_free_dev(sdla_t *card)
-{
- struct net_device* dev;
- volatile wanpipe_common_t *chan;
-
- if (test_and_set_bit(0,&find_free_critical)){
- printk(KERN_INFO "CRITICAL in Find Free\n");
- }
-
- for (dev = card->wandev.dev; dev;
- dev = *((struct net_device **)dev->priv)) {
- chan = dev->priv;
- if (!chan)
- continue;
- if (chan->usedby == API && chan->svc){
- if (!get_atomic_device (dev)){
- if (chan->state != WANSOCK_DISCONNECTED){
- release_device(dev);
- }else{
- clear_bit(0,&find_free_critical);
- return dev;
- }
- }
- }
- clear_bit(0,&find_free_critical);
- return NULL;
-}
-
-/*=====
- * wanpipe_create
- *
- * SOCKET() System call. It allocates a sock structure
- * and adds the socket to the wanpipe_sk_list.
- * Crates AF_WANPIPE socket.
- *=====*/
-
-static int wanpipe_create(struct socket *sock, int protocol)
-{
- struct sock *sk;
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-
- //FIXME: This checks for root user, SECURITY ?
- //if (!capable(CAP_NET_RAW))
- // return -EPERM;
-
- if (sock->type != SOCK_DGRAM && sock->type != SOCK_RAW)
- return -ESOCKTNOSUPPORT;
-
- sock->state = SS_UNCONNECTED;
-
- if ((sk = wanpipe_alloc_socket()) == NULL)
- return -ENOBUFS;
-
- sk->sk_reuse = 1;
- sock->ops = &wanpipe_ops;
- sock_init_data(sock,sk);
-
- sock_reset_flag(sk, SOCK_ZAPPED);
- sk->sk_family = PF_WANPIPE;
- wp_sk(sk)->num = protocol;
- sk->sk_state = WANSOCK_DISCONNECTED;
- sk->sk_ack_backlog = 0;
- sk->sk_bound_dev_if = 0;
-
- atomic_inc(&wanpipe_socks_nr);
-
- /* We must disable interrupts because the ISR
- * can also change the list */
- set_bit(1,&wanpipe_tx_critical);
- write_lock(&wanpipe_sklist_lock);
- sk_add_node(sk, &wanpipe_sklist);
- write_unlock(&wanpipe_sklist_lock);
- clear_bit(1,&wanpipe_tx_critical);
-
- return(0);
-}
-
-/*=====
- * wanpipe_recvmsg
- *
- * Pull a packet from our receive queue and hand it
- * to the user. If necessary we block.
- *=====*/
-
-static int wanpipe_recvmsg(struct kiocb *iocb, struct socket *sock,
- struct msghdr *msg, int len, int flags)
-{
- struct sock *sk = sock->sk;
- struct sk_buff *skb;
- int copied, err=-ENOBUFS;
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-
-
- /*
- * If the address length field is there to be filled in, we fill
- * it in now.
- */
-
- msg->msg_namelen = sizeof(struct wan_sockaddr_ll);
-
- /*
- * Call the generic datagram receiver. This handles all sorts
- * of horrible races and re-entrancy so we can forget about it
- * in the protocol layers.
- *
- * Now it will return ENETDOWN, if device have just gone down,
- * but then it will block.
- */
-
- if (flags & MSG_OOB){
- skb = skb_dequeue(&sk->sk_error_queue);
- }else{
- skb=skb_recv_datagram(sk,flags,1,&err);
- }
- /*
- * An error occurred so return it. Because skb_recv_datagram()
- * handles the blocking we don't see and worry about blocking
- * retries.
- */
-
- if(skb==NULL)
- goto out;
-
- /*
- * You lose any data beyond the buffer you gave. If it worries a
- * user program they can ask the device for its MTU anyway.
- */
-
- copied = skb->len;
- if (copied > len)
- {
- copied=len;
- msg->msg_flags|=MSG_TRUNC;
- }
-
- wanpipe_wakeup_driver(sk);
-
- /* We can't use skb_copy_datagram here */
- err = memcpy_toiovec(msg->msg_iov, skb->data, copied);
- if (err)
- goto out_free;
-
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- sock_recv_timestamp(msg, sk, skb);
-
- if (msg->msg_name)
- memcpy(msg->msg_name, skb->cb, msg->msg_namelen);
-
- /*
- * Free or return the buffer as appropriate. Again this
- * hides all the races and re-entrancy issues from us.
- */
- err = (flags&MSG_TRUNC) ? skb->len : copied;
-
-out_free:
- skb_free_datagram(sk, skb);
-out:
- return err;
-}
-
-/*=====
- * wanpipe_wakeup_driver
- *
- * If socket receive buffer is full and driver cannot
- * pass data up the sock, it sets a packet_block flag.
- * This function check that flag and if sock receive
- * queue has room it kicks the driver BH handler.
- *
- * This way, driver doesn't have to poll the sock
- * receive queue.
- *=====*/
-
-static void wanpipe_wakeup_driver(struct sock *sk)
-{
- struct net_device *dev = NULL;
- wanpipe_common_t *chan=NULL;
-
- dev = dev_get_by_index(sk->sk_bound_dev_if);
- if (!dev)
- return;
-
- dev_put(dev);
-
- if ((chan = dev->priv) == NULL)
- return;
-
- if (atomic_read(&chan->receive_block)){
- if (atomic_read(&sk->sk_rmem_alloc) <
- ((unsigned)sk->sk_rcvbuf * 0.9)) {
- printk(KERN_INFO "wansock: Queuing task for wanpipe\n");
- atomic_set(&chan->receive_block,0);
- wanpipe_queue_tq(&chan->wanpipe_task);
- wanpipe_mark_bh();
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- }
- }
-}
-
-/*=====
- * wanpipe_getname
- *
- * I don't know what to do with this yet.
- * User can use this function to get sock address
- * information. Not very useful for Sangoma's purposes.
- *=====*/
-
-
-static int wanpipe_getname(struct socket *sock, struct sockaddr *uaddr,
- int *uaddr_len, int peer)
-{
- struct net_device *dev;
- struct sock *sk = sock->sk;
- struct wan_sockaddr_ll *sll = (struct wan_sockaddr_ll*)uaddr;
-
- sll->sll_family = AF_WANPIPE;
- sll->sll_ifindex = sk->sk_bound_dev_if;
- sll->sll_protocol = wp_sk(sk)->num;
- dev = dev_get_by_index(sk->sk_bound_dev_if);
- if (dev) {
- sll->sll_hatype = dev->type;
- sll->sll_halen = dev->addr_len;
- memcpy(sll->sll_addr, dev->dev_addr, dev->addr_len);
- } else {
- sll->sll_hatype = 0; /* Bad: we have no ARPHRD_UNSPEC */
- sll->sll_halen = 0;
- }
- *uaddr_len = sizeof(*sll);
-
- dev_put(dev);
-
- return 0;
-}
-
-/*=====
- * wanpipe_notifier
- *
- * If driver turns off network interface, this function
- * will be invoked. Currently I treat it as a
- * call disconnect. More thought should go into this
- * function.
- *
- * FIXME: More thought should go into this function.
- *=====*/
-
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-static int wanpipe_notifier(struct notifier_block *this, unsigned long msg, void *data)
- {
- struct sock *sk;
- hlist_node *node;
- struct net_device *dev = (struct net_device *)data;
-
- sk_for_each(sk, node, &wanpipe_sklist) {
- struct wanpipe_opt *po = wp_sk(sk);
-
- if (!po)
- continue;
- if (dev == NULL)
- continue;
-
- switch (msg) {
- case NETDEV_DOWN:
- case NETDEV_UNREGISTER:
- if (dev->ifindex == sk->sk_bound_dev_if) {
- printk(KERN_INFO "wansock: Device down %s\n", dev->name);
- if (sock_flag(sk, SOCK_ZAPPED)) {
- wanpipe_unlink_driver(sk);
- sk->sk_err = ENETDOWN;
- sk->sk_error_report(sk);
- }
- }
-
- if (msg == NETDEV_UNREGISTER) {
- printk(KERN_INFO "wansock: Unregistering Device: %s\n",
- dev->name);
- wanpipe_unlink_driver(sk);
- sk->sk_bound_dev_if = 0;
- }
- }
- break;
- case NETDEV_UP:
- if (dev->ifindex == sk->sk_bound_dev_if &&
- po->num && !sock_flag(sk, SOCK_ZAPPED)) {
- printk(KERN_INFO "wansock: Registering Device: %s\n",
- dev->name);
- wanpipe_link_driver(dev, sk);
- }
- break;
- }
- return NOTIFY_DONE;
- }
-
-/*=====
- * wanpipe_ioctl
- *
- * Execute a user commands, and set socket options.
- *
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- * FIXME: More thought should go into this function.
- *
- *=====*/
-
-static int wanpipe_ioctl(struct socket *sock, unsigned int cmd, unsigned long arg)
-{
- struct sock *sk = sock->sk;
- int err;
-
- switch(cmd)
- {
- case SIOCGSTAMP:
- return sock_get_timestamp(sk, (struct timeval __user *)arg);
-
- case SIOC_WANPIPE_CHECK_TX:
-
- return atomic_read(&sk->sk_wmem_alloc);
-
- case SIOC_WANPIPE_SOCKET_STATE:
-
- if (sk->sk_state == WANSOCK_CONNECTED)
- return 0;
-
- return 1;
-
- case SIOC_WANPIPE_GET_CALL_DATA:
-
- return get_ioctl_cmd (sk,(void*)arg);
-
- case SIOC_WANPIPE_SET_CALL_DATA:
-
- return set_ioctl_cmd (sk,(void*)arg);
-
- case SIOC_WANPIPE_ACCEPT_CALL:
- case SIOC_WANPIPE_CLEAR_CALL:
- case SIOC_WANPIPE_RESET_CALL:
-
- if ((err=set_ioctl_cmd(sk,(void*)arg)) < 0)
- return err;
-
- err=wanpipe_exec_cmd(sk,cmd,0);
- get_ioctl_cmd(sk,(void*)arg);
- return err;
-
- case SIOC_WANPIPE_DEBUG:
-
- return wanpipe_debug(sk,(void*)arg);
-
- case SIOC_WANPIPE_SET_NONBLOCK:
-

```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- if (sk->sk_state != WANSOCK_DISCONNECTED)
- return -EINVAL;
-
- sock->file->f_flags |= O_NONBLOCK;
- return 0;
-
-#ifdef CONFIG_INET
- case SIOCADDRRT:
- case SIOCDELRT:
- case SIOCDDARP:
- case SIOCGARP:
- case SIOCSARP:
- case SIOCRRARP:
- case SIOCDRARP:
- case SIOCGRRARP:
- case SIOCSRARP:
- case SIOCGIFADDR:
- case SIOCSIFADDR:
- case SIOCGIFBRDADDR:
- case SIOCSIFBRDADDR:
- case SIOCGIFNETMASK:
- case SIOCSIFNETMASK:
- case SIOCGIFDSTADDR:
- case SIOCSIFDSTADDR:
- case SIOCSIFFLAGS:
- return inet_dgram_ops.ioctl(sock, cmd, arg);
-#endif
-
- default:
- return -ENOIOCTLCMD;
- }
- /*NOTREACHED*/
- }
-
-/*=====
- * wanpipe_debug
- *
- * This function will pass up information about all
- * active sockets.
- *
- * FIXME: More thought should go into this function.
- *
- *=====*/
-
-static int wanpipe_debug (struct sock *origsk, void *arg)
- {
- struct sock *sk;
- struct hlist_node *node;
- struct net_device *dev = NULL;
- wanpipe_common_t *chan=NULL;
- int cnt=0, err=0;
- wan_debug_t *dbg_data = (wan_debug_t *)arg;
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
-
- sk_for_each(sk, node, &wanpipe_sklist) {
- wanpipe_opt *wp = wp_sk(sk);
-
- if (sk == origsk){
- continue;
- }
-
- if ((err=put_user(1, &dbg_data->debug[cnt].free)))
- return err;
- if ((err = put_user(sk->sk_state,
- &dbg_data->debug[cnt].state_sk))
- return err;
- if ((err = put_user(sk->sk_rcvbuf,
- &dbg_data->debug[cnt].rcvbuf))
- return err;
- if ((err = put_user(atomic_read(&sk->sk_rmem_alloc),
- &dbg_data->debug[cnt].rmem))
- return err;
- if ((err = put_user(atomic_read(&sk->sk_wmem_alloc),
- &dbg_data->debug[cnt].wmem))
- return err;
- if ((err = put_user(sk->sk_sndbuf,
- &dbg_data->debug[cnt].sndbuf))
- return err;
- if ((err=put_user(sk_count, &dbg_data->debug[cnt].sk_count)))
- return err;
- if ((err=put_user(wp->poll_cnt, &dbg_data->debug[cnt].poll_cnt)))
- return err;
- if ((err = put_user(sk->sk_bound_dev_if,
- &dbg_data->debug[cnt].bound)))
- return err;
-
- if (sk->sk_bound_dev_if) {
- dev = dev_get_by_index(sk->sk_bound_dev_if);
- if (!dev)
- continue;
-
- chan=dev->priv;
- dev_put(dev);
-
- if ((err=put_user(chan->state, &dbg_data->debug[cnt].d_state))
- return err;
- if ((err=put_user(chan->svc, &dbg_data->debug[cnt].svc))
- return err;
-
- if ((err=put_user(atomic_read(&chan->command),
- &dbg_data->debug[cnt].command))
- return err;
-
-
-
-
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".

```
- if (wp){
- sdla_t *card = (sdla_t*)wp->card;
-
- if (card){
- if ((err=put_user(atomic_read(&card->u.x.command_busy),
- &dbg_data->debug[cnt].cmd_busy)))
- return err;
- }
-
- if ((err=put_user(wp->lcn,
- &dbg_data->debug[cnt].lcn)))
- return err;
-
- if (wp->mbox) {
- if ((err=put_user(1, &dbg_data->debug[cnt].mbox)))
- return err;
- }
- }
-
- if ((err=put_user(atomic_read(&chan->receive_block),
- &dbg_data->debug[cnt].rblock)))
- return err;
-
- if (copy_to_user(dbg_data->debug[cnt].name, dev->name, strlen(dev->name)))
- return -EFAULT;
- }
-
- if (++cnt == MAX_NUM_DEBUG)
- break;
- }
- return 0;
-}
-
-/*=====
```

[PATCH] Delete superfluous source file "net/wanrouter/af\_wanpipe.c".