

kgdb 2.0.5

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-01/5105.html>

From: Amit S. Kale (amitkale_at_emsyssoft.com)

Date: 01/20/04

To: Linux Kernel <linux-kernel@vger.kernel.org>, Pavel Machek <pavel@suse.cz>, George Anzinger <george@redhat.com>
Date: Tue, 20 Jan 2004 17:43:29 +0530

Hi,

kgdb 2.0.5 is available at

<http://kgdb.sourceforge.net/kgdb-2/linux-2.6.1-kgdb-2.0.5.tar.bz2>

ChangeLog

2004-01-20 Amit S. Kale <amitkale@emsyssoft.com>

- * Created a ring buffer for kgdb ethernet packets. Several fixes and changes to kgdb on ethernet.

2004-01-20 TimeSys Corporation

- * Fixed a problem with not responding to Ctrl+C during printing of console messages through gdb.

I have pasted below eth.patch for review. When using the ethernet interface, gdb times out several times. It receives packets and junk instead of acks. I see following type of messages out of 8139too.c on the console "eth0:Out-of-sync dirty pointer, 15 vs. 20."

Any comments/suggestions/fixes on this patch are most welcome.

Thanks.

--

Amit Kale

EmSysSoft (<http://www.emsyssoft.com>)

KGDB: Linux Kernel Source Level Debugger (<http://kgdb.sourceforge.net>)

diff -Naurp linux-2.6.1/drivers/net/Kconfig

linux-2.6.1-kgdb-2.0.5-eth/drivers/net/Kconfig

--- linux-2.6.1/drivers/net/Kconfig 2003-11-24 07:02:50.000000000 +0530

+++ linux-2.6.1-kgdb-2.0.5-eth/drivers/net/Kconfig 2004-01-12

19:11:13.000000000 +0530

@@ -187,6 +187,12 @@ config NET_ETHERNET

Note that the answer to this question won't directly affect the kernel: saying N will just cause the configurator to skip all the questions about Ethernet network cards. If unsure, say N.

+

+config KGDB_ETH

+ bool "KGDB: On ethernet"

+ depends on KGDB

+ help

+ Uses ethernet interface for kgdb.

Linux-Kernel: kgdb 2.0.5

```
config MII
    tristate "Generic Media Independent Interface device support"
diff -Naurp linux-2.6.1/drivers/net/kgdb_eth.c
linux-2.6.1-kgdb-2.0.5-eth/drivers/net/kgdb_eth.c
--- linux-2.6.1/drivers/net/kgdb_eth.c 1970-01-01 05:30:00.000000000 +0530
+++ linux-2.6.1-kgdb-2.0.5-eth/drivers/net/kgdb_eth.c 2004-01-20
16:14:22.000000000 +0530
@@ -0,0 +1,704 @@
+/*
+ * Network interface GDB stub
+ *
+ * Copyright (C), 2004 Amit S. Kale
+ *
+ * Written by San Mehat (nettwerk@biodome.org)
+ * Based upon 'gdbserial' by David Grothe (dave@gcom.com)
+ * and Scott Foehner (sfoehner@engr.sgi.com)
+ *
+ * Twiddled for 2.6 by Robert Walsh <rjwalsh@durables.org>
+ * and wangdi <>wangdi@clusterfs.com>.
+ *
+ * Restructured for generic a gdb interface
+ * Reveral changes to make it free of device driver changes.
+ * Added internal buffers for this interface.
+ * by Amit S. Kale <amitkale@emsyssoft.com>
+ * Some cleanups by Pavel Machek <pavel@suse.cz>
+ */
+
+#include <linux/module.h>
+#include <linux/errno.h>
+#include <linux/signal.h>
+#include <linux/sched.h>
+#include <linux/timer.h>
+#include <linux/interrupt.h>
+#include <linux/config.h>
+#include <linux/major.h>
+#include <linux/string.h>
+#include <linux/fcntl.h>
+#include <linux/termios.h>
+#include <linux/kgdb.h>
+#include <linux/if_ether.h>
+#include <linux/netdevice.h>
+#include <linux/etherdevice.h>
+#include <linux/skbuff.h>
+#include <linux/delay.h>
+#include <linux/irq.h>
+#include <linux/inet.h>
+#include <linux/notifier.h>
+#include <net/tcp.h>
+#include <net/udp.h>
+
+#include <asm/system.h>
+#include <asm/io.h>
+#include <asm/segment.h>
+#include <asm/bitops.h>
+#include <asm/system.h>
+#include <asm/atomic.h>
+
+#define GDB_BUF_SIZE 512 /* power of 2, please */
+
+#define CHUNKSIZE 30
+#define MAXINCHUNK (CHUNKSIZE + 8)
```

Linux-Kernel: kgdb 2.0.5

```
+
+static char    kgdb_buf[GDB_BUF_SIZE];
+static int     kgdb_buf_in_inx;
+static atomic_t kgdb_buf_in_cnt;
+static int     kgdb_buf_out_inx;
+
+static unsigned int    kgdb_remoteip = 0;
+static unsigned short kgdb_listenport = 6443;
+static unsigned short kgdb_sendport= 6442;
+static int            kgdb_eth = -1; /* Default tty mode */
+static unsigned char  kgdb_remotemac[6] = {0xff,0xff,0xff,0xff,0xff,0xff};
+static unsigned char  kgdb_localmac[6] = {0xff,0xff,0xff,0xff,0xff,0xff};
+
+static char          kgdbeth_sendbuf[MAXINCHUNK];
+static int           kgdbeth_sendbufchars;
+static irqreturn_t   (*kgdbeth_irqhandler)(int, void *, struct pt_regs *) =
NULL;
+
+struct net_device *kgdb_netdevice = NULL;
+
+/* Indicates dept of recursion for xmitlock hold */
+static int xlockholdcount = 0;
+
+/* kgdb ethernet ring buffers. Increase the space if you get panics in
+ * kgdbeth_alloc_skb.
+ * Status of send_skbs can be known from the field users.
+ * If it's 1 the buffer is free.
+ * If the count 2 or more, the buffer is in use.
+ * Keeping 1 as the initial count prevents kfree_skb from freeing it. */
+#define SEND_BUFLEN 1024
+#define NUM_SENDBUF 128
+static char *send_bufs[NUM_SENDBUF];
+static struct sk_buff *send_skbs[NUM_SENDBUF];
+static struct sk_buff *send_skb;
+static int bufnum;
+
+static char kgdb_netdevname[16];
+
+#ifdef CONFIG_KGDB_CONSOLE
+#error kgdb over ethernet is not yet ready for console messages.
+#endif
+
+/*
+ * Returns next skb from kgdb skbs.
+ * Initializes users field of the skb to 2 so that kfree_skb doesn't attempt
+ * freeing it.
+ * Always call after holding xmitlock of the ethernet device.
+ */
+
+struct sk_buff *kgdbeth_alloc_skb(int size)
+{
+    struct sk_buff *skb;
+    u8              *data;
+    int i;
+
+    i = bufnum;
+
+    do {
+        skb = send_skbs[i];
+
+        if (atomic_read(&skb->users) == 1) {
+            bufnum = i;
```

Linux-Kernel: kgdb 2.0.5

```
+             i = -1;
+             break;
+         }
+         i = (i + 1) % NUM_SENDBUF;
+     } while (i != bufnum);
+     if (i == bufnum) {
+         panic("kgdb ethernet buffer overflow\n");
+     }
+     data = (u8 *) (send_bufs[bufnum]);
+
+     size = SKB_DATA_ALIGN(size);
+     if (size + sizeof(struct skb_shared_info) > SEND_BUFLLEN)
+         panic("kgdb ethernet buffer too short for this request");
+
+     memset(skb, 0, offsetof(struct sk_buff, truesize));
+     skb->truesize = size + sizeof(struct sk_buff);
+     atomic_set(&skb->users, 2);
+     skb->head = data;
+     skb->data = data;
+     skb->tail = data;
+     skb->end = data + size;
+
+     atomic_set(&(skb_shinfo(skb)->dataref), 1);
+     skb_shinfo(skb)->nr_frags = 0;
+     skb_shinfo(skb)->tso_size = 0;
+     skb_shinfo(skb)->tso_segs = 0;
+     skb_shinfo(skb)->frag_list = NULL;
+
+     return skb;
+}
+
+/* Holds xmitlock of the ethernet device
+ * Recursive calls allowed */
+static void kgdbeth_holdxlock(void)
+{
+     if (spin_is_locked(&kgdb_netdevice->xmit_lock)) {
+         if (kgdb_netdevice->xmit_lock_owner == smp_processor_id()) {
+             goto gotit;
+         }
+     }
+     spin_lock(&kgdb_netdevice->xmit_lock);
+     kgdb_netdevice->xmit_lock_owner = smp_processor_id();
+
+gotit:
+     xlockholdcount++;
+}
+
+/* releases xmitlock of the ethernet device
+ * Recursive calls allowed */
+static void kgdbeth_relxlock(void)
+{
+     if (--xlockholdcount) {
+         kgdb_netdevice->xmit_lock_owner = -1;
+         spin_unlock(&kgdb_netdevice->xmit_lock);
+     }
+}
+
+/*
+ * Get a char if available, return -1 if nothing available.
+ * Empty the receive buffer first, then look at the interface hardware.
+ */
+static int
```

Linux-Kernel: kgdb 2.0.5

```
+read_char(void)
+{
+    /* intr routine has queued chars */
+    if (atomic_read(&kgdb_buf_in_cnt) != 0) {
+        int chr;
+
+        chr = kgdb_buf[kgdb_buf_out_inx++];
+        kgdb_buf_out_inx &= (GDB_BUF_SIZE - 1);
+        atomic_dec(&kgdb_buf_in_cnt);
+        return chr;
+    }
+
+    return -1; /* no data */
+}
+
+/*
+ * Wait until the interface can accept a char, then write it.
+ */
+static void
+write_buffer(char *buf, int len)
+{
+    int                total_len, eth_len, ip_len, udp_len;
+    struct in_device   *in_dev;
+    struct sk_buff     *skb;
+    struct udphdr     *udph;
+    struct iphdr       *iph;
+    struct ethhdr     *eth;
+
+    if (!(in_dev = (struct in_device *) kgdb_netdevice->ip_ptr)) {
+        panic("No in_device available for interface!\n");
+    }
+
+    if (!(in_dev->ifa_list)) {
+        panic("No interface address set for interface!\n");
+    }
+    kgdbeth_holdxlock();
+
+    udp_len = len + sizeof(struct udphdr);
+    ip_len = eth_len = udp_len + sizeof(struct iphdr);
+    total_len = eth_len + ETH_HLEN;
+
+    skb = kgdbeth_alloc_skb(total_len);
+
+    skb_reserve(skb, total_len - 1);
+
+    memcpy(skb->data, (unsigned char *) buf, len);
+    skb->len += len;
+
+    udph = (struct udphdr *) skb_push(skb, sizeof(*udph));
+    udph->source = htons(kgdb_listenport);
+    udph->dest   = htons(kgdb_sendport);
+    udph->len    = htons(udp_len);
+    udph->check  = 0;
+
+    iph = (struct iphdr *) skb_push(skb, sizeof(*iph));
+    iph->version = 4;
+    iph->ihl     = 5;
+    iph->tos     = 0;
+    iph->tot_len = htons(ip_len);
+    iph->id     = 0;
+    iph->frag_off = 0;
+    iph->tttl    = 64;
```

Linux-Kernel: kgdb 2.0.5

```
+     iph->protocol = IPPROTO_UDP;
+     iph->check     = 0;
+     iph->saddr     = in_dev->ifa_list->ifa_address;
+     iph->daddr     = kgdb_remoteip;
+     iph->check     = ip_fast_csum((unsigned char *)iph, iph->ihl);
+
+     eth = (struct ethhdr *) skb_push(skb, ETH_HLEN);
+     eth->h_proto = htons(ETH_P_IP);
+     memcpy(eth->h_source, kgdb_localmac, kgdb_netdevice->addr_len);
+     memcpy(eth->h_dest, kgdb_remotemac, kgdb_netdevice->addr_len);
+
+     kgdb_netdevice->hard_start_xmit(skb, kgdb_netdevice);
+     if (atomic_read(&skb->users) != 1) {
+         BUG();
+     }
+     kgdbeth_relxlock();
+}
+
+static void kgdbeth_flush(void)
+{
+     if (!kgdbeth_sendbufchars) {
+         return;
+     }
+     write_buffer(kgdbeth_sendbuf, kgdbeth_sendbufchars);
+     kgdbeth_sendbufchars = 0;
+}
+
+static void kgdbeth_write_char(int chr)
+{
+     if (kgdbeth_sendbufchars == MAXINCHUNK) {
+         kgdbeth_flush();
+     }
+     kgdbeth_sendbuf[kgdbeth_sendbufchars++] = chr;
+}
+
+/*
+ * In the interrupt state the target machine will not respond to any
+ * arp requests, so handle them here.
+ */
+
+static void
+kgdb_eth_reply_arp(void)
+{
+     if (send_skb) {
+         kgdbeth_holdxlock();
+         kgdb_netdevice->hard_start_xmit(send_skb, kgdb_netdevice);
+         send_skb = NULL;
+         kgdbeth_relxlock();
+     }
+}
+
+static int
+make_arp_request(struct sk_buff *skb)
+{
+     struct arphdr *arp;
+     unsigned char *arp_ptr;
+     int type = ARPOP_REPLY;
+     int ptype = ETH_P_ARP;
+     u32 sip, tip;
+     unsigned char *sha, *tha;
+     struct in_device *in_dev = (struct in_device *) kgdb_netdevice->ip_ptr;
```

Linux-Kernel: kgdb 2.0.5

```
+
+ /* No arp on this interface */
+
+ if (kgdb_netdevice->flags & IFF_NOARP) {
+     return 0;
+ }
+
+ if (!pskb_may_pull(skb, (sizeof(struct arphdr) +
+                          (2 * kgdb_netdevice->addr_len) +
+                          (2 * sizeof(u32)))))) {
+     return 0;
+ }
+
+ skb->h.raw = skb->nh.raw = skb->data;
+ arp = skb->nh.arph;
+
+ if ((arp->ar_hrd != htons(ARPHRD_ETHER) &&
+      arp->ar_hrd != htons(ARPHRD_IEEE802)) ||
+     arp->ar_pro != htons(ETH_P_IP)) {
+     return 0;
+ }
+
+ /* Understand only these message types */
+
+ if (arp->ar_op != htons(ARPOP_REQUEST)) {
+     return 0;
+ }
+
+ /* Extract fields */
+
+ arp_ptr = (unsigned char *)(arp+1);
+ sha = arp_ptr;
+ arp_ptr += kgdb_netdevice->addr_len;
+ memcpy(&sip, arp_ptr, 4);
+ arp_ptr += 4;
+ tha = arp_ptr;
+ arp_ptr += kgdb_netdevice->addr_len;
+ memcpy(&tip, arp_ptr, 4);
+
+ if (tip != in_dev->ifa_list->ifa_address) {
+     return 0;
+ }
+
+ if (kgdb_remoteip != sip) {
+     return 0;
+ }
+
+ /*
+  * Check for bad requests for 127.x.x.x and requests for multicast
+  * addresses.  If this is one such, delete it.
+  */
+
+ if (LOOPBACK(tip) || MULTICAST(tip)) {
+     return 0;
+ }
+
+ /* reply to the ARP request */
+
+ kgdbeth_holdxlock();
+ if (send_skb) {
+     /* Get rid of any previous replies to ARP request. We hope
+      * that regular reply to ARP by network layers would have gone
+      * out by now. */
```

Linux-Kernel: kgdb 2.0.5

```
+         kfree_skb(send_skb);
+     }
+     send_skb = kgdbeth_alloc_skb(sizeof(struct arphdr) +
+                               2 * (kgdb_netdevice->addr_len + 4) +
+                               LL_RESERVED_SPACE(kgdb_netdevice));
+     kgdbeth_relxlock();
+
+     skb_reserve(send_skb, LL_RESERVED_SPACE(kgdb_netdevice));
+     send_skb->nh.raw = send_skb->data;
+     arp = (struct arphdr *) skb_put(send_skb, sizeof(struct arphdr) + 2 *
(kgdb_netdevice->addr_len + 4));
+     send_skb->dev = kgdb_netdevice;
+     send_skb->protocol = htons(ETH_P_ARP);
+
+     /* Fill the device header for the ARP frame */
+
+     if (kgdb_netdevice->hard_header &&
+         kgdb_netdevice->hard_header(send_skb, kgdb_netdevice, ptype,
+                                     kgdb_remotemac, kgdb_localmac,
+                                     send_skb->len) < 0) {
+         kfree_skb(send_skb);
+         return 0;
+     }
+
+     /*
+      * Fill out the arp protocol part.
+      *
+      * we only support ethernet device type,
+      * which (according to RFC 1390) should always equal 1 (Ethernet).
+      */
+
+     arp->ar_hrd = htons(kgdb_netdevice->type);
+     arp->ar_pro = htons(ETH_P_IP);
+
+     arp->ar_hln = kgdb_netdevice->addr_len;
+     arp->ar_pln = 4;
+     arp->ar_op = htons(type);
+
+     arp_ptr=(unsigned char *) (arp + 1);
+
+     memcpy(arp_ptr, kgdb_netdevice->dev_addr, kgdb_netdevice->addr_len);
+     arp_ptr += kgdb_netdevice->addr_len;
+     memcpy(arp_ptr, &tip, 4);
+     arp_ptr += 4;
+     memcpy(arp_ptr, kgdb_localmac, kgdb_netdevice->addr_len);
+     arp_ptr += kgdb_netdevice->addr_len;
+     memcpy(arp_ptr, &sip, 4);
+     return 0;
+ }
+
+ /*
+  * Accept an skbuff from net_device layer and add the payload onto
+  * kgdb buffer
+  *
+  * When the kgdb stub routine read_char() is called it draws characters
+  * out of the buffer until it is empty and then polls the hardware.
+  *
+  * Return value of NET_RX_DROP means skb was used by this function.
+  * NET_RX_SUCCESS indicates this function didn't use it.
+  *
+  * Be prepared to respond to ARP requests. Let the caller also handle
```

Linux–Kernel: kgdb 2.0.5

```
+ * them if debugger is not active. We'll respond to the ARP request when
+ * a debugging session begins. It's necessary to respond to the request as
the
+ * debugging session may begin even before kernel has a chance finish the
+ * response.
+ */
+int
+kgdb_net_interrupt(struct sk_buff *skb)
+{
+    unsigned char    chr;
+    struct iphdr     *iph = (struct iphdr*)skb->data;
+    struct udphdr    *udph= (struct udphdr*)(skb->data+(iph->ihl<<2));
+    unsigned char    *data = (unsigned char *) udph + sizeof(struct udphdr);
+    int              len;
+    int              i;
+
+    if (!kgdb_initialized || !kgdb_netdevice) {
+        goto out;
+    }
+    if (skb->protocol == __constant_htons(ETH_P_ARP) && !send_skb) {
+        make_arp_request(skb);
+        goto out;
+    }
+    if (iph->protocol != IPPROTO_UDP ||
+        be16_to_cpu(udph->dest) != kgdb_listenport)
+        goto out;
+
+    kgdb_sendport = be16_to_cpu(udph->source);
+
+    len = (be16_to_cpu(iph->tot_len) -
+          (sizeof(struct udphdr) + sizeof(struct iphdr)));
+
+    for (i = 0; i < len; i++) {
+        chr = *data++;
+        if (chr == 3)
+        {
+            if (!atomic_read(&debugger_active)) {
+                breakpoint();
+            }
+            continue;
+        }
+        if (atomic_read(&kgdb_buf_in_cnt) >= GDB_BUF_SIZE) {
+            /* buffer overflow, clear it */
+            kgdb_buf_in_inx = 0;
+            atomic_set(&kgdb_buf_in_cnt, 0);
+            kgdb_buf_out_inx = 0;
+            break;
+        }
+        kgdb_buf[kgdb_buf_in_inx++] = chr;
+        kgdb_buf_in_inx &= (GDB_BUF_SIZE - 1);
+        atomic_inc(&kgdb_buf_in_cnt) ;
+    }
+    return NET_RX_DROP;
+
+out:
+    if (atomic_read(&debugger_active))
+        return NET_RX_DROP;
+    return NET_RX_SUCCESS;
+}
+/*
+ * Initializes ethernet interface to kgdb.
```

Linux-Kernel: kgdb 2.0.5

```
+ * Searches the device list and finds the device specified on kernel command
+ * line.
+ * Finds the irq handler for the device and saves its reference.
+ * Initializes kgdbeth data structures.
+ */
+
+int
+kgdbeth_hook(void)
+{
+    extern void kgdb_respond_ok(void);
+    struct irqaction *ia_ptr;
+    int i;
+
+    sprintf(kgdb_netdevname, "eth%d", kgdb_eth);
+
+    for (kgdb_netdevice = dev_base;
+         kgdb_netdevice != NULL;
+         kgdb_netdevice = kgdb_netdevice->next) {
+        if (strncmp(kgdb_netdevice->name, kgdb_netdevname, IFNAMSIZ) == 0) {
+            break;
+        }
+    }
+    if (!kgdb_netdevice) {
+        printk("kgdbeth: Unable to find interface %s\n",
+              kgdb_netdevname);
+        return -ENODEV;
+    }
+    if (!(kgdb_netdevice->flags & IFF_UP)) {
+        return -EINVAL;
+    }
+    ia_ptr = irq_desc[kgdb_netdevice->irq].action;
+    while (ia_ptr) {
+        if (!strncmp(kgdb_netdevname, ia_ptr->name, IFNAMSIZ)) {
+            kgdbeth_irqhandler = ia_ptr->handler;
+            break;
+        }
+        ia_ptr = ia_ptr->next;
+    }
+    if (!kgdbeth_irqhandler) {
+        printk("kgdbeth: Interface %s doesn't have an interrupt"
+              " handler cannot use it\n", kgdb_netdevname);
+        return -EINVAL;
+    }
+    for (i = 0; i < NUM_SENDBUF; i++) {
+        send_skbs[i] = kmalloc(sizeof(struct sk_buff), GFP_KERNEL);
+        send_bufs[i] = kmalloc(SEND_BUFLLEN, GFP_KERNEL);
+        if (!send_skbs[i] || !send_bufs[i]) {
+            printk("kgdbeth: not enough memory\n");
+            return -ENOMEM;
+        }
+        atomic_set(&(send_skbs[i]->users), 1);
+    }
+
+    return 0;
+}
+
+/*
+ * kgdbeth_read_char
+ *
+ * This is a GDB stub routine. It waits for a character from the
+ * ethernet interface and then returns it.
```

Linux-Kernel: kgdb 2.0.5

```
+ */
+static int
+kgdbeth_read_char(void)
+{
+    int    chr;
+
+    while ((chr = read_char()) < 0) {
+        if (send_skb) {
+            kgdb_eth_reply_arp();
+        }
+        (*kgdbeth_irqhandler)(kgdb_netdevice->irq,
+                               (void *)kgdb_netdevice, 0);
+    }
+    return chr;
+}
+
+/*
+ * Hold onto the xmitlock and keep holding till the session ends.
+ * Disable device irq and keep it disabled till this session ends.
+ * Respond to last arp request we have received. It may not have be
+ * responded yet.
+ */
+static void kgdbeth_begin_session(void) {
+    kgdbeth_holdxlock();
+    disable_irq(kgdb_netdevice->irq);
+    kgdb_eth_reply_arp();
+}
+
+static void kgdbeth_end_session(void)
+{
+    enable_irq(kgdb_netdevice->irq);
+    kgdbeth_relxlock();
+}
+
+struct kgdb_serial kgdbeth_serial = {
+    .chunksize = CHUNKSIZE,
+    .read_char = kgdbeth_read_char,
+    .write_char = kgdbeth_write_char,
+    .hook = kgdbeth_hook,
+    .flush = kgdbeth_flush,
+    .begin_session = kgdbeth_begin_session,
+    .end_session = kgdbeth_end_session,
+};
+
+static int __init parse_hw_addr(char **ptr, unsigned char *addr,
+                                unsigned char delimiter)
+{
+    int i = 0;
+
+    while(**ptr != delimiter)
+    {
+        unsigned int c;
+
+        if (sscanf(*ptr, "%x", &c) != 1) {
+            return 1;
+        }
+        addr[i++] = c;
+        do
+            (*ptr)++;
+        while(**ptr != delimiter) && (*(ptr-1) != ':');
+        if (i > 6)
+            return -EINVAL;
+    }
+}
```

Linux-Kernel: kgdb 2.0.5

```
+     }
+     if (i != 6)
+         return -EINVAL;
+     return 0;
+ }
+
+int kgdbeth_thread(void *data)
+{
+     struct net_device *ndev = (struct net_device *)data;
+     daemonize("kgdbeth");
+     while (!ndev->ip_ptr) {
+         schedule();
+     }
+     debugger_entry();
+     return 0;
+}
+
+int kgdbeth_event(struct notifier_block * self, unsigned long val, void *
data)
+{
+     if (strcmp(((struct net_device *)data)->name, kgdb_netdevname)) {
+         goto out;
+     }
+     if (val!= NETDEV_UP)
+         goto out;
+     kernel_thread(kgdbeth_thread, data, CLONE_KERNEL);
+
+out:
+     return NOTIFY_DONE;
+}
+static struct notifier_block nb = {
+     .notifier_call = kgdbeth_event,
+};
+
+/*
+ * Syntax for this cmdline option is
+ * kgdbeth=interfacenum,localmac,listenport,remoteip,remotemac
+ */
+
+static int __init kgdbeth_opt(char *str)
+{
+     char ipaddrstr[16];
+     char *ipaddrptr = ipaddrstr;
+     extern int register_netdevice_notifier(struct notifier_block *nb);
+
+     if (register_netdevice_notifier(&nb)) {
+         printk("KGDB_ETH: couldn't register notifier\n");
+         return 0;
+     }
+
+     /* interfacenum */
+     if (*str < '0' || *str > '9')
+         goto errout;
+     kgdb_eth = *str - '0';
+     str++;
+     if (*str != ',')
+         goto errout;
+     str++;
+
+     /* localmac */
+     if (parse_hw_addr(&str, kgdb_localmac, ','))
```

Linux-Kernel: kgdb 2.0.5

```

+         goto errout;
+     str++;
+
+     /* port */
+     kgdb_listenport = simple_strtoul(str, &str, 10);
+
+     if (*str != ',')
+         goto errout;
+     str++;
+
+     /* remoteip */
+     while (*str != ',') {
+         if (!*str)
+             goto errout;
+         *ipaddrptr = *str;
+         str++;
+         ipaddrptr++;
+     }
+     str++;
+     *ipaddrptr = '\0';
+     kgdb_remoteip = in_aton(ipaddrstr);
+
+     /* remotemac */
+     if (parse_hw_addr(&str, kgdb_remotemac, '\0'))
+         goto errout;
+
+     kgdb_serial = &kgdbeth_serial;
+     return 1;
+errout:
+     printk("Invalid syntax for option kgdbeth=\n");
+     return 0;
+}
+
+__setup("kgdbeth=", kgdbeth_opt);
diff -Naurp linux-2.6.1/drivers/net/Makefile
linux-2.6.1-kgdb-2.0.5-eth/drivers/net/Makefile
--- linux-2.6.1/drivers/net/Makefile      2003-11-24 07:01:27.000000000 +0530
+++ linux-2.6.1-kgdb-2.0.5-eth/drivers/net/Makefile      2004-01-12
19:11:13.000000000 +0530
@@ -32,6 +32,8 @@ obj-$(CONFIG_BMAC) += bmac.o

obj-$(CONFIG_OAKNET) += oaknet.o 8390.o

+obj-$(CONFIG_KGDB_ETH) += kgdb_eth.o
+
obj-$(CONFIG_DGRS) += dgrs.o
obj-$(CONFIG_RCPCI) += rcpci.o
obj-$(CONFIG_VORTEX) += 3c59x.o
diff -Naurp linux-2.6.1/net/core/dev.c
linux-2.6.1-kgdb-2.0.5-eth/net/core/dev.c
--- linux-2.6.1/net/core/dev.c      2004-01-10 11:01:50.000000000 +0530
+++ linux-2.6.1-kgdb-2.0.5-eth/net/core/dev.c      2004-01-20 14:27:38.000000000
+0530
@@ -1380,7 +1380,6 @@ static void sample_queue(unsigned long d
}
#endif

-
/**
 * netif_rx      -      post buffer to the network code
 * @skb: buffer to post

```

Linux-Kernel: kgdb 2.0.5

```
@@ -1404,6 +1403,15 @@ int netif_rx(struct sk_buff *skb)
    int this_cpu;
    struct softnet_data *queue;
    unsigned long flags;
+
+   int ret;
+   int kgdb_net_interrupt(struct sk_buff *skb);
+
+#ifdef CONFIG_KGDB_ETH
+   /* See if kgdb_eth wants this packet */
+   if ((ret = kgdb_net_interrupt(skb)) == NET_RX_DROP) {
+       return ret;
+   }
+#endif

    if (!skb->stamp.tv_sec)
        do_gettimeofday(&skb->stamp);
diff -Naurp linux-2.6.1/net/core/skbuff.c
linux-2.6.1-kgdb-2.0.5-eth/net/core/skbuff.c
--- linux-2.6.1/net/core/skbuff.c      2003-11-24 07:02:36.000000000 +0530
+++ linux-2.6.1-kgdb-2.0.5-eth/net/core/skbuff.c      2004-01-20 16:10:42.000000000
+0530
@@ -55,6 +55,7 @@
#include <linux/rtnetlink.h>
#include <linux/init.h>
#include <linux/highmem.h>
+#include <linux/debugger.h>

#include <net/protocol.h>
#include <net/dst.h>
@@ -126,6 +127,11 @@ struct sk_buff *alloc_skb(unsigned int s
{
    struct sk_buff *skb;
    u8 *data;
+
+   struct sk_buff *kgdbeth_alloc_skb(int size);
+
+   if (atomic_read(&debugger_active)) {
+       return kgdbeth_alloc_skb(size);
+   }

    /* Get the HEAD */
    skb = kmem_cache_alloc(skbuff_head_cache,
```

-
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org

More majordomo info at <http://vger.kernel.org/majordomo-info.html>

Please read the FAQ at <http://www.tux.org/lkml/>