

## [KGDB PATCH][2/7] Serial updates, take 2

**Source:** <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-02/7479.html>

---

**From:** Tom Rini (*trini\_at\_kernel.crashing.org*)

**Date:** 02/27/04

Date: Fri, 27 Feb 2004 14:25:48 -0700

To: Kernel Mailing List <linux-kernel@vger.kernel.org>, Pavel Machek <pavel@suse.cz>, amit@gate.c

The following is my second attempt at making the serial driver more robust.

This has the following changes:

- We don't need `kgdb_might_be_resumed` or `kgdb_killed_or_detached`.
- Don't try and look for a connection in `put_packet`, after we've tried to put a packet. Instead, when we receive a packet, GDB has connected. Also make `put_packet` look at the char it reads, and handle a new packet in the middle (or `^C+packet`).
- Remove `ok_packet()`, excessive, IMHO.
- In `kgdb8250_interrupt`, only check if this is the right line, and if so, call `kgdb_schedule_breakpoint()`. This ends up killing off a lot of (incomplete due to my not testing SMP) locking.
- Remove a duplicate extern of `kgdb_connected`.

```
diff -zrupN linux-2.6.3+nothing/arch/ppc/8260_io/uart.c linux-2.6.3+config+serial/arch/ppc/8260_io/uart.c
--- linux-2.6.3+nothing/arch/ppc/8260_io/uart.c 2004-02-27 12:16:14.397164756 -0700
+++ linux-2.6.3+config+serial/arch/ppc/8260_io/uart.c 2004-02-27 12:16:13.765307745 -0700
@@ -396,16 +396,10 @@ static _INLINE_ void receive_chars(ser_i
#ifdef CONFIG_KGDB
    if (info->state->smc_scc_num == KGDB_SER_IDX) {
        while (i-- > 0) {
- if (*cp == 0x03) {
+ if (*cp == 0x03 || *cp == '$') {
            breakpoint();
            return;
        }
- if (*cp == '$') {
- atomic_set(&kgdb_might_be_resumed, 1);
- breakpoint();
- atomic_set(&kgdb_might_be_resumed, 0);
- return;
- }
            cp++;
        }
        bdp->cbd_sc |= BD_SC_EMPTY;
diff -zrupN linux-2.6.3+nothing/drivers/serial/kgdb_8250.c
linux-2.6.3+config+serial/drivers/serial/kgdb_8250.c
--- linux-2.6.3+nothing/drivers/serial/kgdb_8250.c 2004-02-27 12:16:14.557128556 -0700
```

## Linux-Kernel: [KGDB PATCH][2/7] Serial updates, take 2

```

+++ linux-2.6.3+config+serial/drivers/serial/kgdb_8250.c 2004-02-27 12:16:13.000000000 -0700
@@ -17,7 +17,6 @@
#include <linux/config.h>
#include <linux/kernel.h>
#include <linux/init.h>
-#include <linux/spinlock.h>
#include <linux/kgdb.h>
#include <linux/interrupt.h>
#include <linux/tty.h>
@@ -166,32 +165,15 @@ read_data_bfr(void)
/*
 * Get a char if available, return -1 if nothing available.
 * Empty the receive buffer first, then look at the interface hardware.
- *
- * Locking here is a bit of a problem. We MUST not lock out communication
- * if we are trying to talk to gdb about a kgdb entry. ON the other hand
- * we can loose chars in the console pass thru if we don't lock. It is also
- * possible that we could hold the lock or be waiting for it when kgdb
- * NEEDS to talk. Since kgdb locks down the world, it does not need locks.
- * We do, of course have possible issues with interrupting a uart operation,
- * but we will just depend on the uart status to help keep that straight.
 */

-static spinlock_t uart_interrupt_lock = SPIN_LOCK_UNLOCKED;
-#ifdef CONFIG_SMP
-extern spinlock_t kgdb_spinlock;
-#endif
-
int
kgdb_get_debug_char(void)
{
    int retchr;
    unsigned long flags;
    local_irq_save(flags);
-#ifdef CONFIG_SMP
- if (!spin_is_locked(&kgdb_spinlock)) {
- spin_lock(&uart_interrupt_lock);
- }
-#endif
+
    /* intr routine has q'd chars */
    if (atomic_read(&kgdb8250_buf_in_cnt) != 0) {
        retchr = kgdb8250_buf[kgdb8250_buf_out_inx++];
@@ -205,11 +187,6 @@ kgdb_get_debug_char(void)
    } while (retchr < 0);

out:
-#ifdef CONFIG_SMP
- if (!spin_is_locked(&kgdb_spinlock)) {
- spin_unlock(&uart_interrupt_lock);
- }

```

```

-#endif
    local_irq_restore(flags);

    return retchr;
@@ -217,72 +194,17 @@ out:

/*
 * This is the receiver interrupt routine for the GDB stub.
- * It will receive a limited number of characters of input
- * from the gdb host machine and save them up in a buffer.
- *
- * When kgdb_get_debug_char() is called it
- * draws characters out of the buffer until it is empty and
- * then reads directly from the serial port.
- *
- * We do not attempt to write chars from the interrupt routine
- * since the stubs do all of that via kgdb_put_debug_char() which
- * writes one byte after waiting for the interface to become
- * ready.
- *
- * The debug stubs like to run with interrupts disabled since,
- * after all, they run as a consequence of a breakpoint in
- * the kernel.
- *
- * Perhaps someone who knows more about the tty driver than I
- * care to learn can make this work for any low level serial
- * driver.
+ * All that we need to do is verify that the interrupt happened on the
+ * line we're in charge of. If this is true, schedule a breakpoint and
+ * return.
 */
static irqreturn_t
kgdb8250_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{
- int chr, iir;
- unsigned long flags;
-
    if (irq != gdb_async_info.line)
        return IRQ_NONE;

- /* If we get an interrupt, then KGDB is trying to connect. */
- if (!kgdb_connected) {
- kgdb_schedule_breakpoint();
- return IRQ_HANDLED;
- }
-
- local_irq_save(flags);
- spin_lock(&uart_interrupt_lock);
-
- do {
- chr = read_data_bfr();

```

## Linux-Kernel: [KGDB PATCH][2/7] Serial updates, take 2

```
- iir = serial_inb(kgdb8250_port + (UART_IIR << reg_shift));
- if (chr < 0)
- continue;
-
- /* Check whether gdb sent interrupt */
- if (chr == 3) {
- breakpoint();
- continue;
- }
-
- if (atomic_read(&kgdb8250_buf_in_cnt) >= GDB_BUF_SIZE) {
- /* buffer overflow, clear it */
- kgdb8250_buf_in_inx = 0;
- atomic_set(&kgdb8250_buf_in_cnt, 0);
- kgdb8250_buf_out_inx = 0;
- break;
- }
-
- kgdb8250_buf[kgdb8250_buf_in_inx++] = chr;
- kgdb8250_buf_in_inx &= (GDB_BUF_SIZE - 1);
- atomic_inc(&kgdb8250_buf_in_cnt);
- } while (iir & UART_IIR_RDI);
-
- spin_unlock(&uart_interrupt_lock);
- local_irq_restore(flags);
-
+ kgdb_schedule_breakpoint();
  return IRQ_HANDLED;
}
```

```
diff -zrupN linux-2.6.3+nothing/include/linux/kgdb.h linux-2.6.3+config+serial/include/linux/kgdb.h
--- linux-2.6.3+nothing/include/linux/kgdb.h 2004-02-27 12:16:14.000000000 -0700
+++ linux-2.6.3+config+serial/include/linux/kgdb.h 2004-02-27 12:16:13.000000000 -0700
@@ -36,10 +36,6 @@ extern volatile int kgdb_connected;
#endif
```

```
extern atomic_t kgdb_setting_breakpoint;
-extern atomic_t kgdb_killed_or_detached;
-extern atomic_t kgdb_might_be_resumed;
-
-extern volatile int kgdb_connected;
```

```
extern struct task_struct *kgdb_usethread, *kgdb_contthread;
```

```
diff -zrupN linux-2.6.3+nothing/kernel/kgdb.c linux-2.6.3+config+serial/kernel/kgdb.c
--- linux-2.6.3+nothing/kernel/kgdb.c 2004-02-27 12:16:14.000000000 -0700
+++ linux-2.6.3+config+serial/kernel/kgdb.c 2004-02-27 12:16:13.000000000 -0700
@@ -15,6 +15,7 @@
 * Copyright (C) 2002-2004 Timesys Corporation
 * Copyright (C) 2003-2004 Amit S. Kale
 * Copyright (C) 2004 Pavel Machek <pavel@suse.cz>
```

```

+ * Copyright (C) 2004 Tom Rini <trini@kernel.crashing.org>
+ *
+ * Restructured KGDB for 2.6 kernels.
+ * thread support, support for multiple processors, support for ia-32(x86)
@@ -87,8 +88,6 @@ static const char hexchars[] = "01234567
static spinlock_t slavecpulocks[KGDB_MAX_NO_CPUS];
static volatile int procindebug[KGDB_MAX_NO_CPUS];
atomic_t kgdb_setting_breakpoint;
-atomic_t kgdb_killed_or_detached;
-atomic_t kgdb_might_be_resumed;
struct task_struct *kgdb_uthread, *kgdb_contthread;

int debugger_step;
@@ -212,8 +211,10 @@ static void get_packet(char *buffer)
    char ch;

    do {
- /* wait around for the start character, ignore all other characters */
- while ((ch = (kgdb_serial->read_char() & 0x7f)) != '$') ;
+ /* wait around for the start character, ignore all other
+ * characters */
+ while ((ch = (kgdb_serial->read_char() & 0x7f)) != '$')
+ ; /* Spin. */
        kgdb_connected = 1;
        checksum = 0;
        xmitsum = -1;
@@ -249,27 +250,22 @@ static void get_packet(char *buffer)
    * Send the packet in buffer.
    * Check for gdb connection if asked for.
    */
- static void put_packet(char *buffer, int checkconnect)
+ static void put_packet(char *buffer)
    {
        unsigned char checksum;
        int count;
        char ch;
- static char gdbseq[] = "$Hc-1#09";
- int i;
- int send_count;

        /* $<packet info>#<checksum>. */
- do {
+ while (1) {
            kgdb_serial->write_char('$');
            checksum = 0;
            count = 0;
- send_count = 0;

            while ((ch = buffer[count])) {
                kgdb_serial->write_char(ch);
                checksum += ch;

```

```

        count++;
- send_count++;
    }

    kgdb_serial->write_char('#');
@@ -278,30 +274,27 @@ static void put_packet(char *buffer, int
    if (kgdb_serial->flush)
        kgdb_serial->flush();

- i = 0;
- while ((ch = kgdb_serial->read_char()) == gdbseq[i++] &&
- checkconnect) {
- if (!gdbseq[i]) {
- kgdb_serial->write_char('+');
- if (kgdb_serial->flush)
- kgdb_serial->flush();
- breakpoint();
-
- /*
- * GDB is available now.
- * Retransmit this packet.
- */
- break;
- }
- }
- if (checkconnect && ch == 3) {
- kgdb_serial->write_char('+');
+ /* Now see what we get in reply. */
+ ch = kgdb_serial->read_char();
+
+ if (ch == 3)
+ ch = kgdb_serial->read_char();
+
+ /* If we get an ACK, we are done. */
+ if (ch == '+')
+ return;
+
+ /* If we get the start of another packet, this means
+ * that GDB is attempting to reconnect. We will NAK
+ * the packet being sent, and stop trying to send this
+ * packet. */
+ if (ch == '$') {
+ kgdb_serial->write_char('-');
+         if (kgdb_serial->flush)
+             kgdb_serial->flush();
- breakpoint();
+ return;
    }
- } while ((ch & 0x7f) != '+');
-
+ }

```

```

}

static int get_char(char *addr, unsigned char *data)
@@ -427,11 +420,6 @@ static inline void error_packet(char *pk
    pkt[3] = '\0';
}

-static void ok_packet(char *pkt)
- {
-     strcpy(pkt, "OK");
- }
-
static char *pack_threadid(char *pkt, threadref * id)
{
    char *limit;
@@ -502,7 +490,8 @@ void kgdb_wait(struct pt_regs *regs)
    procindebug[processor] = 1;
    current->thread.debuggerinfo = regs;

- /* Wait till master processor goes completely into the debugger. FIXME: this looks racy */
+ /* Wait till master processor goes completely into the debugger.
+ * FIXME: this looks racy */
    while (!procindebug[atomic_read(&debugger_active) - 1]) {
        int i = 10; /* an arbitrary number */
@@ -701,17 +690,7 @@ int kgdb_handle_exception(int exVector,
    /* Master processor is completely in the debugger */
    kgdb_post_master_code(linux_regs, exVector, err_code);

- if (atomic_read(&kgdb_killed_or_detached) &&
-     atomic_read(&kgdb_might_be_resumed)) {
-     get_packet(remcom_in_buffer);
-     if (remcom_in_buffer[0] == 'H' && remcom_in_buffer[1] == 'c') {
-         remove_all_break();
-         atomic_set(&kgdb_killed_or_detached, 0);
-         ok_packet(remcom_out_buffer);
-     } else
-         return 1;
-     } else {
-
+ if (kgdb_connected) {
+     /* reply to host that an exception has occurred */
+     ptr = remcom_out_buffer;
+     *ptr++ = 'T';
@@ -721,11 +700,9 @@ int kgdb_handle_exception(int exVector,
    int_to_threadref(&thref, shadow_pid(current->pid));
    ptr = pack_threadid(ptr, &thref);
    *ptr++ = ';';
- *ptr = '\0';
- }

```

```

- put_packet(remcom_out_buffer, 0);
- kgdb_connected = 1;
+ put_packet(remcom_out_buffer);
+ }

    kgdb_usethread = current;
    kgdb_usethreadid = shadow_pid(current->pid);
@@ -741,6 +718,11 @@ int kgdb_handle_exception(int exVector,

    switch (remcom_in_buffer[0]) {
    case '?':
+ /* We know that this packet is only sent
+ * during initial connect. So to be safe,
+ * we clear out our breakpoints now incase
+ * GDB is reconnecting. */
+ remove_all_break();
        remcom_out_buffer[0] = 'S';
        remcom_out_buffer[1] = hexchars[signo >> 4];
        remcom_out_buffer[2] = hexchars[signo % 16];
@@ -798,7 +780,7 @@ int kgdb_handle_exception(int exVector,
        else {
            gdb_regs_to_regs(gdb_regs, (struct pt_regs *)
                current->thread.debuggerinfo);
- ok_packet(remcom_out_buffer);
+ strcpy(remcom_out_buffer, "OK");
        }

        break;
@@ -838,10 +820,10 @@ int kgdb_handle_exception(int exVector,
        if ((error = remove_all_break()) < 0) {
            error_packet(remcom_out_buffer, error);
        } else {
- ok_packet(remcom_out_buffer);
+ strcpy(remcom_out_buffer, "OK");
            kgdb_connected = 0;
        }
- put_packet(remcom_out_buffer, 0);
+ put_packet(remcom_out_buffer);
        goto default_handle;

    case 'k':
@@ -947,11 +929,10 @@ int kgdb_handle_exception(int exVector,
        }
        kgdb_usethread = thread;
        kgdb_usethreadid = threadid;
- ok_packet(remcom_out_buffer);
+ strcpy(remcom_out_buffer, "OK");
        break;

    case 'c':
- atomic_set(&kgdb_killed_or_detached, 0);

```

## Linux-Kernel: [KGDB PATCH][2/7] Serial updates, take 2

```

ptr = &remcom_in_buffer[2];
kgdb_hex2long(&ptr, &threadid);
if (!threadid) {
@@ -966,7 +947,7 @@ int kgdb_handle_exception(int exVector,
    }
    kgdb_contthread = thread;
}
- ok_packet(remcom_out_buffer);
+ strcpy(remcom_out_buffer, "OK");
    break;
}
break;
@@ -977,7 +958,7 @@ int kgdb_handle_exception(int exVector,
    kgdb_hex2long(&ptr, &threadid);
    thread = getthread(linux_regs, threadid);
    if (thread)
- ok_packet(remcom_out_buffer);
+ strcpy(remcom_out_buffer, "OK");
    else
        error_packet(remcom_out_buffer, -EINVAL);
    break;
@@ -1018,7 +999,7 @@ int kgdb_handle_exception(int exVector,
    }

    if (error == 0)
- ok_packet(remcom_out_buffer);
+ strcpy(remcom_out_buffer, "OK");
    else
        error_packet(remcom_out_buffer, error);

@@ -1039,7 +1020,7 @@ int kgdb_handle_exception(int exVector,
    } /* switch */

    /* reply to the request */
- put_packet(remcom_out_buffer, 0);
+ put_packet(remcom_out_buffer);
}

kgdb_exit:
@@ -1063,7 +1044,6 @@ int kgdb_handle_exception(int exVector,
    }

    /* Free debugger_active */
- atomic_set(&kgdb_killed_or_detached, 1);
    atomic_set(&debugger_active, 0);
    local_irq_restore(flags);

@@ -1132,12 +1112,6 @@ void kgdb_entry(void)
    /* Free debugger_active */
    atomic_set(&debugger_active, 0);

```

## Linux-Kernel: [KGDB PATCH][2/7] Serial updates, take 2

```
- /* This flag is used, if gdb has detached and wants to start
- * another session
- */
- atomic_set(&kgdb_killed_or_detached, 1);
- atomic_set(&kgdb_might_be_resumed, 0);
-
-     for (i = 0; i < MAX_BREAKPOINTS; i++)
-         kgdb_break[i].state = bp_disabled;

@@ -1222,7 +1196,7 @@ void kgdb_console_write(struct console *
-     *bufptr = '\0';
-     s += wcount;

- put_packet(kgdbconbuf, 1);
+ put_packet(kgdbconbuf);

-     }
-     local_irq_restore(flags);

--
Tom Rini
http://gate.crashing.org/~trini/
-
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/
```