

kgdb cleanups

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2004-01/2361.html>

From: Pavel Machek (*pavel_at_ucw.cz*)

Date: 01/09/04

Date: Fri, 9 Jan 2004 19:38:26 +0100

To: kernel list <linux-kernel@vger.kernel.org>, Andrew Morton <akpm@zip.com.au>

Hi!

No real code changes, but cleanups all over the place. What about applying?

Ouch and arch–dependend code is moved to kernel/kgdb.c. I'll probably do x86–64 version so that is rather important.

Pavel

--- tmp/linux/Documentation/i386/kgdb/kgdb.txt 2004-01-09 19:04:43.000000000 +0100

+++ linux/Documentation/i386/kgdb/kgdb.txt 2003-12-25 15:11:19.000000000 +0100

@@ -1,242 +1,9 @@

-Last edit: <20030806.1637.12>

-This file has information specific to the i386 kgdb option. Other

-platforms with the kgdb option may behave in a similar fashion.

-

-New features:

=====

-20030806.1557.37

-This version was made against the 2.6.0–test2 kernel. We have made the

-following changes:

-

--- The getthread() code in the stub calls find_task_by_pid(). It fails

- if we are early in the bring up such that the pid arrays have yet to

- be allocated. We have added a line to kernel/pid.c to make

- "kgdb_pid_init_done" true once the arrays are allocated. This way the

- getthread() code knows not to call. This is only used by the thread

- debugging stuff and threads will not yet exist at this point in the

- boot.

-

--- For some reason, gdb was not asking for a new thread list when the

- "info thread" command was given. We changed to the newer version of

- the thread info command and gdb now seems to ask when needed. Result,

- we now get all threads in the thread list.

-

--- We now respond to the ThreadExtraInfo request from gdb with the thread

Linux-Kernel: kgdb cleanups

- name from task_struct .comm. This then appears in the thread list.
- Thoughts on additional options for this are welcome. Things such as
- "has BKL" and "Preempted" come to mind. I think we could have a flag
- word that could enable different bits of info here.
-
- We now honor, sort of, the C and S commands. These are continue and
- single set after delivering a signal. We ignore the signal and do the
- requested action. This only happens when we told gdb that a signal
- was the reason for entry, which is only done on memory faults. The
- result is that you can now continue into the Oops.
-
- We changed the -g to -gdwarf-2. This seems to be the same as -ggdb,
- but it is more exact on what language to use.
-
- We added two dwarf2 include files and a bit of code at the end of
- entry.S. This does not yet work, so it is disabled. Still we want to
- keep track of the code and "maybe" someone out there can fix it.
-
- Randy Dunlap sent some fix ups for this file which are now merged.
-
- Hugh Dickins sent a fix to a bit of code in traps.c that prevents a
- compiler warning if CONFIG_KGDB is off (now who would do that :).
-
- Andrew Morton sent a fix for the serial driver which is now merged.
-
- Andrew also sent a change to the stub around the cpu managment code
- which is also merged.
-
- Andrew also sent a patch to make "f" as well as "g" work as SysRq
- commands to enter kgdb, merged.
-
- If CONFIG_KGDB and CONFIG_DEBUG_SPINLOCKS are both set we added a
- "who" field to the spinlock data struct. This is filled with
- "current" when ever the spinlock succeeds. Useful if you want to know
- who has the lock.
-
- _ And last, but not least, we fixed the "get_cu" macro to properly get
- the current value of "current".
-
- New features:
- =====
- 20030505.1827.27
- We are starting to align with the sourceforge version, at least in
- commands. To this end, the boot command string to start kgdb at
- boot time has been changed from "kgdb" to "gdb".
-
- Andrew Morton sent a couple of patches which are now included as follows:
- 1.) We now return a flag to the interrupt handler.
- 2.) We no longer use smp_num_cpus (a conflict with the lock meter).
- 3.) And from William Lee Irwin III <wli@holomorphy.com> code to make
- sure high-mem is set up before we attempt to register our interrupt

Linux–Kernel: kgdb cleanups

- handler.
- We now include asm/kgdb.h from config.h so you will most likely never
- have to include it. It also 'NULLS' the kgdb macros you might have in
- your code when CONFIG_KGDB is not defined. This allows you to just
- turn off CONFIG_KGDB to turn off all the kgdb_ts() calls and such.
- This include is conditioned on the machine being an x86 so as to not
- mess with other archs.
-
- 20020801.1129.03
- This is currently the version for the 2.4.18 (and beyond?) kernel.
-
- We have several new "features" beginning with this version:
-
- 1.) Kgdb now syncs the "other" CPUs with a cross–CPU NMI. No more
- waiting and it will pull that guy out of an IRQ off spin lock :)
-
- 2.) We doctored up the code that tells where a task is waiting and
- included it so that the "info thread" command will show a bit more
- than "schedule()". Try it...
-
- 3.) Added the ability to call a function from gdb. All the standard gdb
- issues apply, i.e. if you hit a breakpoint in the function, you are
- not allowed to call another (gdb limitation, not kgdb). To help
- this capability we added a memory allocation function. Gdb does not
- return this memory (it is used for strings that you pass to that function
- you are calling from gdb) so we fixed up a way to allow you to
- manually return the memory (see below).
-
- 4.) Kgdb time stamps (kgdb_ts()) are enhanced to expand what was the
- interrupt flag to now also include the preemption count and the
- "in_interrupt" info. The flag is now called "with_pif" to indicate
- the order, preempt_count, in_interrupt, flag. The preempt_count is
- shifted left by 4 bits so you can read the count in hex by dropping
- the low order digit. In_interrupt is in bit 1, and the flag is in
- bit 0.
-
- 5.) The command: "p kgdb_info" is now expanded and prints something
- like:
- (gdb) p kgdb_info
- \$2 = {used_malloc = 0, called_from = 0xc0107506, entry_tsc = 67468627259,
- errcode = 0, vector = 3, print_debug_info = 0, hold_on_sstep = 1,
- cpus_waiting = { {task = 0xc027a000, pid = 32768, hold = 0,
- regs = 0xc027bf84}, {task = 0x0, pid = 0, hold = 0, regs = 0x0} } }
-
- Things to note here: a.) used_malloc is the amount of memory that
- has been malloc'ed to do calls from gdb. You can reclaim this
- memory like this: "p kgdb_info.used_malloc=0" Cool, huh? b.)
- cpus_waiting is now "sized" by the number of CPUs you enter at
- configure time in the kgdb configure section. This is NOT used
- anywhere else in the system, but it is "nice" here. c.) The task's
- "pid" is now in the structure. This is the pid you will need to use

Linux–Kernel: kgdb cleanups

- to decode to the thread id to get gdb to look at that thread.
- Remember that the "info thread" command prints a list of threads
- wherein it numbers each thread with its reference number followed
- by the thread's pid. Note that the per–CPU idle threads actually
- have pids of 0 (yes, there is more than one pid 0 in an SMP system).
- To avoid confusion, kgdb numbers these threads with numbers beyond
- the MAX_PID. That is why you see 32768 and above.
-
- 6.) A subtle change, we now provide the complete register set for tasks
- that are active on the other CPUs. This allows better trace back on
- those tasks.
-
- And, let's mention what we could not fix. Back–trace from all but the
- thread that we trapped will, most likely, have a bogus entry in it.
- The problem is that gdb does not recognize the entry code for
- functions that use "current" near (at all?) the entry. The compiler
- is putting the "current" decode as the first two instructions of the
- function where gdb expects to find %ebp changing code. Back trace
- also has trouble with interrupt frames. I am talking with Daniel
- Jacobowitz about some way to fix this, but don't hold your breath.
-
- 20011220.0050.35
- Major enhancement with this version is the ability to hold one or more
- CPUs in an SMP system while allowing the others to continue. Also, by
- default only the current CPU is enabled on single–step commands (please
- note that gdb issues single–step commands at times other than when you
- use the si command).
-
- Another change is to collect some useful information in
- a global structure called "kgdb_info". You should be able to just:
-
- p kgdb_info
-
- although I have seen cases where the first time this is done gdb just
- prints the first member but prints the whole structure if you then enter
- CR (carriage return or enter). This also works:
-
- p *&kgdb_info
-
- Here is a sample:
- (gdb) p kgdb_info
- \$4 = {called_from = 0xc010732c, entry_tsc = 32804123790856, errcode = 0,
- vector = 3, print_debug_info = 0}
-
- "Called_from" is the return address from the current entry into kgdb.
- Sometimes it is useful to know why you are in kgdb, for example, was
- it an NMI or a real breakpoint? The simple way to interrogate this
- return address is:
-
- l *0xc010732c
-

Linux–Kernel: kgdb cleanups

- which will print the surrounding few lines of source code.
-
- "Entry_tsc" is the CPU TSC on entry to kgdb (useful to compare to the kgdb_ts entries).
-
- "errcode" and "vector" are other entry parameters which may be helpful on some traps.
-
- "print_debug_info" is the internal debugging kgdb print enable flag. Yes, you can modify it.
-
- In SMP systems kgdb_info also includes the "cpus_waiting" structure and "hold_on_step":
-
- (gdb) p kgdb_info
- \$7 = {called_from = 0xc0112739, entry_tsc = 1034936624074, errcode = 0, vector = 2, print_debug_info = 0, hold_on_sstep = 1, cpus_waiting = {{ task = 0x0, hold = 0, regs = 0x0}, {task = 0xc71b8000, hold = 0, regs = 0xc71b9f70}, {task = 0x0, hold = 0, regs = 0x0}, {task = 0x0, hold = 0, regs = 0x0}, {task = 0x0, hold = 0, regs = 0x0}, {task = 0x0, hold = 0, regs = 0x0}, {task = 0x0, hold = 0, regs = 0x0}, {task = 0x0, hold = 0, regs = 0x0}, {task = 0x0, hold = 0, regs = 0x0}, {task = 0x0, hold = 0, regs = 0x0}}}
-
- "Cpus_waiting" has an entry for each CPU other than the current one that has been stopped. Each entry contains the task_struct address for that CPU, the address of the regs for that task and a hold flag. All these have the proper typing so that, for example:
-
- p *kgdb_info.cpus_waiting[1].regs
-
- will print the registers for CPU 1.
-
- "Hold_on_sstep" is a new feature with this version and comes up set or true. What this means is that whenever kgdb is asked to single-step all other CPUs are held (i.e. not allowed to execute). The flag applies to all but the current CPU and, again, can be changed:
-
- p kgdb_info.hold_on_sstep=0
-
- restores the old behavior of letting all CPUs run during single-stepping.
-
- Likewise, each CPU has a "hold" flag, which if set, locks that CPU out of execution. Note that this has some risk in cases where the CPUs need to communicate with each other. If kgdb finds no CPU available on exit, it will push a message thru gdb and stay in kgdb. Note that it is legal to hold the current CPU as long as at least one CPU can execute.
-
- 20010621.1117.09
- This version implements an event queue. Events are signaled by calling a function in the kgdb stub and may be examined from gdb. See EVENTS below for details. This version also tightens up the interrupt and SMP

Linux–Kernel: kgdb cleanups

- handling to not allow interrupts on the way to kgdb from a breakpoint
- trap. It is fine to allow these interrupts for user code, but not
- system debugging.

–

Version

=====

- This version of the kgdb package was developed and tested on
- kernel version 2.4.16. It will not install on any earlier kernels.
- It is possible that it will continue to work on later versions
- of 2.4 and then versions of 2.5 (I hope).
- +This version of the kgdb package was developed and tested on kernel
- +version 2.6.0. It is possible that it will continue to work on later
- +versions of 2.6.

Debugging Setup

@@ -314,7 +81,7 @@

set up after any serial drivers, it is possible that this command will work when the control–C will not.

- Save and exit the Xconfig program. Then do "make clean" , "make dep"
- +Save and exit the Xconfig program. Then do "make clean"
- and "make bzImage" (or whatever target you want to make). This gets the kernel compiled with the "–g" option set -- necessary for debugging.

```
diff -ur -x '.dep*' -x '.hdep*' -x '.*[oas]' -x '*~' -x '#*' -x '*CVS*' -x '*.orig' -x '*.rej' -x '*.old' -x '.menu*'
-x asm -x local.h -x System.map -x autoconf.h -x compile.h -x version.h -x .version -x defkeymap.c -x
uni_hash.tbl -x zImage -x vmlinux -x vmlinuz -x TAGS -x bootsect -x '*RCS*' -x conmakehash -x map
-x build -x build -x configure -x '*target*' -x '*.flags' -x '*.bak' -x '*.cmd'
tmp/linux/Documentation/i386/kgdb/kgdbeth.txt linux/Documentation/i386/kgdb/kgdbeth.txt
--- tmp/linux/Documentation/i386/kgdb/kgdbeth.txt 2004-01-09 19:04:43.000000000 +0100
+++ linux/Documentation/i386/kgdb/kgdbeth.txt 2003-12-25 14:55:58.000000000 +0100
@@ -33,8 +33,8 @@
```

```
kgdbeth=DEVICENUM
kgdbeth_remoteip=HOSTIPADDR
- kgdbeth_remotemac=REMOTEMAC
- kgdbeth_localmac=LOCALMAC
+ kgdbeth_remotemac=HOSTMAC
+ kgdbeth_localmac=TARGETMAC
```

kgdbeth=DEVICENUM sets the ethernet device number to listen on for debugging packets. e.g. kgdbeth=0 listens on eth0.

@@ -43,10 +43,10 @@

Only packets originating from this IP address will be accepted by the debugger. e.g. kgdbeth_remoteip=192.168.2.2

- kgdbeth_remotemac=REMOTEMAC sets the ethernet address of the HOST machine.
- +kgdbeth_remotemac=HOSTMAC sets the ethernet address of the HOST machine.
- e.g. kgdbeth_remotemac=00:07:70:12:4E:F5

Linux–Kernel: kgdb cleanups

–kgdbeth_localmac=LOCALMAC sets the ethernet address of the TARGET machine.
+kgdbeth_localmac=TARGETMAC sets the ethernet address of the TARGET machine.
e.g. kgdbeth_localmac=00:10:9F:18:21:3C

You can also set the following command–line option on the TARGET kernel:

```
--- tmp/linux/arch/i386/kernel/entry.S 2004–01–09 19:04:43.000000000 +0100  
+++ linux/arch/i386/kernel/entry.S 2003–12–27 17:08:29.000000000 +0100  
@@ –71,51 +71,6 @@  
# location or end up with a return address pointing at the  
# location, we don't need a correct call frame for it.
```

```
–#if 0  
–  
–#include <linux/dwarf2–lang.h>  
–/*  
– * The register numbers as known by gdb  
– */  
–#define _EAX 0  
–#define _ECX 1  
–#define _EDX 2  
–#define _EBX 3  
–#define _ESP 4  
–#define _EBP 5  
–#define _ESI 6  
–#define _EDI 7  
–#define _PC 8  
–#define _EIP 8  
–#define _PS 9  
–#define _EFLAGS 9  
–#define _CS 10  
–#define _SS 11  
–#define _DS 12  
–#define _ES 13  
–#define _FS 14  
–#define _GS 15  
–  
– CFI_preamble(c1,_PC,1,1)  
– CFA_define_reference(_ESP,OLDESP)  
– CFA_define_offset(_EIP,EIP)  
– CFA_define_offset(_EBX,EBX)  
– CFA_define_offset(_ECX,ECX)  
– CFA_define_offset(_EDX,EDX)  
– CFA_define_offset(_ESI,ESI)  
– CFA_define_offset(_EDI,EDI)  
– CFA_define_offset(_EBP,EBP)  
– CFA_define_offset(_EAX,EAX)  
– CFA_define_offset(_EFLAGS,EFLAGS)  
– CFA_define_offset(_CS,CS)  
– CFA_define_offset(_DS,DS)  
– CFA_define_offset(_ES,ES)
```

Linux–Kernel: kgdb cleanups

```
– CFI_postamble(c1)
–
– FDE_preamble(c1,f1,ret_from_intr,(divide_error – ret_from_intr))
– FDE_postamble(f1)
–#endif
–
EBX = 0x00
ECX = 0x04
EDX = 0x08
@@ –366,19 +321,6 @@
    testw $_TIF_ALLWORK_MASK, %cx # current–>work
    jne syscall_exit_work
restore_all:
–#ifdef CONFIG_TRAP_BAD_SYSCALL_EXITS
– movl EFLAGS(%esp), %eax # mix EFLAGS and CS
– movb CS(%esp), %al
– testl $(VM_MASK | 3), %eax
– jz resume_kernelX # returning to kernel or vm86–space
–
– cmpl $0,TI_PRE_COUNT(%ebx) # non–zero preempt_count ?
– jz resume_kernelX
–
– int $3
–
–resume_kernelX:
–#endif
    RESTORE_ALL

    # perform work that needs to be done immediately before resumption
--- tmp/linux/arch/i386/kernel/kgdb_stub.c 2004–01–09 19:04:43.000000000 +0100
+++ linux/arch/i386/kernel/kgdb_stub.c 2003–12–28 21:40:20.000000000 +0100
@@ –1,4 +1,8 @@
/*
+ * Kernel gdb stub, arch–dependend part
+ *
+ * Copyright (c) 2000 VERITAS Software Corporation.
+ * Copyright (c) 2003 Pavel Machek <pavel@suse.cz>
+ *
+ * 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
@@ –12,98 +16,7 @@
+ *
+ */
–/*
– * Copyright (c) 2000 VERITAS Software Corporation.
– *
– */
–/*****
– * Header: remcom.c,v 1.34 91/03/09 12:29:49 glenne Exp $
– *
```

Linux–Kernel: kgdb cleanups

```
– * Module name: remcom.c $
– * Revision: 1.34 $
– * Date: 91/03/09 12:29:49 $
– * Contributor: Lake Stevens Instrument Division$
– *
– * Description: low level support for gdb debugger. $
– *
– * Considerations: only works on target hardware $
– *
– * Written by: Glenn Engel $
– * Updated by: David Grothe <dave@gcom.com>
– * Updated by: Robert Walsh <rjwalsh@durables.org>
– * Updated by: wangdi <>wangdi@clusterfs.com>
– * ModuleState: Experimental $
– *
– * NOTES: See Below $
– *
– * Modified for 386 by Jim Kingdon, Cygnus Support.
– * Compatibility with 2.1.xx kernel by David Grothe <dave@gcom.com>
– *
– * Changes to allow auto initialization. All that is needed is that it
– * be linked with the kernel and a break point (int 3) be executed.
– * The header file <asm/kgdb.h> defines BREAKPOINT to allow one to do
– * this. It should also be possible, once the interrupt system is up, to
– * call putDebugChar("+"). Once this is done, the remote debugger should
– * get our attention by sending a ^C in a packet. George Anzinger
– * <george@mvista.com>
– * Integrated into 2.2.5 kernel by Tigran Aivazian <tigran@sco.com>
– * Added thread support, support for multiple processors,
– * support for ia–32(x86) hardware debugging.
– * Amit S. Kale ( akale@veritas.com )
– *
– * Modified to support debugging over ethernet by Robert Walsh
– * <rjwalsh@durables.org> and wangdi <>wangdi@clusterfs.com>, based on
– * code by San Mehat.
– *
– *
– * To enable debugger support, two things need to happen. One, a
– * call to set_debug_traps() is necessary in order to allow any breakpoints
– * or error conditions to be properly intercepted and reported to gdb.
– * Two, a breakpoint needs to be generated to begin communication. This
– * is most easily accomplished by a call to breakpoint(). Breakpoint()
– * simulates a breakpoint by executing an int 3.
– *
– *
– * *****
– *
– * The following gdb commands are supported:
– *
– * command function Return value
– *
– * g return the value of the CPU registers hex data or ENN
```

Linux–Kernel: kgdb cleanups

```
– * G set the value of the CPU registers OK or ENN
– *
– * mAA..AA,LLLL Read LLLL bytes at address AA..AA hex data or ENN
– * MAA..AA,LLLL: Write LLLL bytes at address AA.AA OK or ENN
– *
– * c Resume at current address SNN ( signal NN)
– * cAA..AA Continue at address AA..AA SNN
– *
– * s Step one instruction SNN
– * sAA..AA Step one instruction from AA..AA SNN
– *
– * k kill
– *
– * ? What was the last signal ? SNN (signal NN)
– *
– * All commands and responses are sent with a packet which includes a
– * checksum. A packet consists of
– *
– * $<packet info>#<checksum>.
– *
– * where
– * <packet info> :: <characters representing the command or response>
– * <checksum> :: < two hex digits computed as modulo 256 sum of <packetinfo>>
– *
– * When a packet is received, it is first acknowledged with either '+' or '-'.
– * '+' indicates a successful transfer. '-' indicates a failed transfer.
– *
– * Example:
– *
– * Host: Reply:
– * $m0,10#2a+$00010203040506070809101112131415#42
– *
– *****/
–#define KGDB_VERSION "<20030915.1651.33>"
+#define KGDB_VERSION "2.6.0"
#include <linux/config.h>
#include <linux/types.h>
#include <asm/string.h> /* for strcpy */
@@ -121,41 +34,11 @@
#include <linux/inet.h>
#include <linux/kallsyms.h>

_/* *****/
– *
– * external low–level support routines
– */
–typedef void (*Function) (void); /* pointer to a function */
–
–/* Thread reference */
–typedef unsigned char threadref[8];
–
```

Linux–Kernel: kgdb cleanups

```
–extern int tty_putDebugChar(int); /* write a single character */
–extern int tty_getDebugChar(void); /* read and return a single char */
–extern void tty_flushDebugChar(void); /* flush pending characters */
–extern int eth_putDebugChar(int); /* write a single character */
–extern int eth_getDebugChar(void); /* read and return a single char */
–extern void eth_flushDebugChar(void); /* flush pending characters */
–extern void kgdb_eth_set_trapmode(int);
–extern void kgdb_eth_reply_arp(void); /*send arp request */
–extern volatile int kgdb_eth_is_initializing;
–
–
_/******
_/* BUFMAX defines the maximum number of characters in inbound/outbound buffers*/
_/* at least NUMREGBYTES*2 are needed for register packets */
_/* Longer buffer is needed to list all threads */
_#define BUFMAX 400
–
–char *kgdb_version = KGDB_VERSION;
–
_/* debug > 0 prints ill–formed commands in valid packets & checksum errors */
–int debug_regs = 0; /* set to non–zero to print registers */
–
_/* filled in by an external module */
–char *gdb_module_offsets;
+#if __GNUC__ < 3
+#error Sorry, your GCC is too old to work with kgdb. (It works with 3.3.2)
+#endif

–static const char hexchars[] = "0123456789abcdef";
+#include "../kernel/kgdb.c"

_/* Number of bytes of registers. */
#define NUMREGBYTES 64
@@ –184,135 +67,11 @@
    _GS /* 15 */
};

_/****** ASSEMBLY CODE MACROS *****/
_/*
– * Put the error code here just in case the user cares.
– * Likewise, the vector number here (since GDB only gets the signal
– * number through the usual means, and that's not very specific).
– * The called_from is the return address so he can tell how we entered kgdb.
– * This will allow him to separate out the various possible entries.
– */
_#define REMOTE_DEBUG 0 /* set != to turn on printing (also available in info) */
–
_#define PID_MAX PID_MAX_DEFAULT
–
_#ifdef CONFIG_SMP
–void smp_send_nmi_allbutself(void);
```

Linux–Kernel: kgdb cleanups

```
–#define IF_SMP(x) x
–#undef MAX_NO_CPUS
–#ifndef CONFIG_NO_KGDB_CPUS
–#define CONFIG_NO_KGDB_CPUS 2
–#endif
–#if CONFIG_NO_KGDB_CPUS > NR_CPUS
–#define MAX_NO_CPUS NR_CPUS
–#else
–#define MAX_NO_CPUS CONFIG_NO_KGDB_CPUS
–#endif
–#define hold_init hold_on_sstep: 1,
–#define MAX_CPU_MASK (unsigned long)((1LL << MAX_NO_CPUS) – 1LL)
–#define NUM_CPUS num_online_cpus()
–#else
–#define IF_SMP(x)
–#define hold_init
–#undef MAX_NO_CPUS
–#define MAX_NO_CPUS 1
–#define NUM_CPUS 1
–#endif
–#define NOCPU (struct task_struct *)0xbad1fbad
–/* *INDENT–OFF* */
–struct kgdb_info {
– int used_malloc;
– void *called_from;
– long long entry_tsc;
– int errcode;
– int vector;
– int print_debug_info;
–#ifdef CONFIG_SMP
– int hold_on_sstep;
– struct {
– volatile struct task_struct *task;
– int pid;
– int hold;
– struct pt_regs *regs;
– } cpus_waiting[MAX_NO_CPUS];
–#endif
–} kgdb_info = {hold_init print_debug_info:REMOTE_DEBUG, vector:–1};
–
–/* *INDENT–ON* */
–
–#define used_m kgdb_info.used_malloc
–/*
– * This is little area we set aside to contain the stack we
– * need to build to allow gdb to call functions. We use one
– * per cpu to avoid locking issues. We will do all this work
– * with interrupts off so that should take care of the protection
– * issues.
– */
–#define LOOKASIDE_SIZE 200 /* should be more than enough */
```

Linux–Kernel: kgdb cleanups

```
–#define MALLOC_MAX 200 /* Max malloc size */
–struct {
– unsigned int esp;
– int array[LOOKASIDE_SIZE];
–} fn_call_lookaside[MAX_NO_CPUS];
–
–static int trap_cpu;
static unsigned int OLD_esp;

–#define END_OF_LOOKASIDE &fn_call_lookaside[trap_cpu].array[LOOKASIDE_SIZE]
#define IF_BIT 0x200
#define TF_BIT 0x100

–#define MALLOC_ROUND 8–1
–
–static char malloc_array[MALLOC_MAX];
–IF_SMP(static void to_gdb(const char *mess));
–void *
–malloc(int size)
–{
–
– if (size <= (MALLOC_MAX – used_m)) {
– int old_used = used_m;
– used_m += ((size + MALLOC_ROUND) & (~MALLOC_ROUND));
– return &malloc_array[old_used];
– } else {
– return NULL;
– }
–}
–
–/*
– * I/O dispatch functions...
– * Based upon kgdb_eth, either call the ethernet
– * handler or the serial one..
– */
–void
–putDebugChar(int c)
–{
– if (kgdb_eth == –1) {
– tty_putDebugChar(c);
– } else {
– eth_putDebugChar(c);
– }
–}
–
–int
–getDebugChar(void)
–{
– if (kgdb_eth == –1) {
– return tty_getDebugChar();
– } else {
```

Linux-Kernel: kgdb cleanups

```
- return eth_getDebugChar();
- }
-}
-
-void
-flushDebugChar(void)
-{
- if (kgdb_eth == -1) {
- tty_flushDebugChar();
- } else {
- eth_flushDebugChar();
- }
-}

/*
 * Gdb calls functions by pushing arguments, including a return address
@@ -345,7 +104,7 @@
 */
extern asmlinkage void fn_call_stub(void);
extern asmlinkage void fn_rtn_stub(void);
-/* *INDENT-OFF* */
+
+__asm__( "fn_rtn_stub:\n\t"
+        "movl %eax,%esp\n\t"
+        "fn_call_stub:\n\t"
@@ -359,213 +118,11 @@"
+        "popl %ebx\n\t"
+        "popl %ecx\n\t"
+        "iret \n\t");
-/* *INDENT-ON* */
+
+#define gdb_i386vector kgdb_info.vector
+#define gdb_i386errcode kgdb_info.errcode
-#define waiting_cpus kgdb_info.cpus_waiting
-#define remote_debug kgdb_info.print_debug_info
-#define hold_cpu(cpu) kgdb_info.cpus_waiting[cpu].hold
-/* gdb locks */
-
-#ifdef CONFIG_SMP
-static int in_kgdb_called;
-static spinlock_t waitlocks[MAX_NO_CPUS] =
- { [0 ... MAX_NO_CPUS - 1] = SPIN_LOCK_UNLOCKED };
-/*
- * The following array has the thread pointer of each of the "other"
- * cpus. We make it global so it can be seen by gdb.
- */
-volatile int in_kgdb_entry_log[MAX_NO_CPUS];
-volatile struct pt_regs *in_kgdb_here_log[MAX_NO_CPUS];
-/*
-static spinlock_t continuelocks[MAX_NO_CPUS];
-*/
```

Linux–Kernel: kgdb cleanups

```
–spinlock_t kgdb_spinlock = SPIN_LOCK_UNLOCKED;
–/* waiters on our spinlock plus us */
–static atomic_t spinlock_waiters = ATOMIC_INIT(1);
–static int spinlock_count = 0;
–static int spinlock_cpu = 0;
–/*
– * Note we use nested spin locks to account for the case where a break
– * point is encountered when calling a function by user direction from
– * kgdb. Also there is the memory exception recursion to account for.
– * Well, yes, but this lets other cpus thru too. Lets add a
– * cpu id to the lock.
– */
–#define KGDB_SPIN_LOCK(x) if( spinlock_count == 0 || \
– spinlock_cpu != smp_processor_id()){ \
– atomic_inc(&spinlock_waiters); \
– while (! spin_trylock(x)) { \
– in_kgdb(&regs); \
– } \
– atomic_dec(&spinlock_waiters); \
– spinlock_count = 1; \
– spinlock_cpu = smp_processor_id(); \
– }else{ \
– spinlock_count++; \
– }
–#define KGDB_SPIN_UNLOCK(x) if( --spinlock_count == 0) spin_unlock(x)
–#else
–unsigned kgdb_spinlock = 0;
–#define KGDB_SPIN_LOCK(x) --*x
–#define KGDB_SPIN_UNLOCK(x) ++*x
–#endif
–
–int
–hex(char ch)
–{
– if ((ch >= 'a') && (ch <= 'f'))
– return (ch - 'a' + 10);
– if ((ch >= '0') && (ch <= '9'))
– return (ch - '0');
– if ((ch >= 'A') && (ch <= 'F'))
– return (ch - 'A' + 10);
– return (-1);
–}

–/* scan for the sequence $<data>#<checksum> */
–void
–getpacket(char *buffer)
–{
– unsigned char checksum;
– unsigned char xmitsum;
– int i;
– int count;
```

```

– char ch;

– do {
– /* wait around for the start character, ignore all other characters */
– while ((ch = (getDebugChar() & 0x7f)) != '$') ;
– checksum = 0;
– xmitcsum = -1;
–
– count = 0;
–
– /* now, read until a # or end of buffer is found */
– while (count < BUFMAX) {
– ch = getDebugChar() & 0x7f;
– if (ch == '#')
– break;
– checksum = checksum + ch;
– buffer[count] = ch;
– count = count + 1;
– }
– buffer[count] = 0;
–
– if (ch == '#') {
– xmitcsum = hex(getDebugChar() & 0x7f) << 4;
– xmitcsum += hex(getDebugChar() & 0x7f);
– if ((remote_debug) && (checksum != xmitcsum)) {
– printk
– ("bad checksum. My count = 0x%x, sent=0x%x. buf=%s\n",
– checksum, xmitcsum, buffer);
– }
–
– if (checksum != xmitcsum)
– putDebugChar('-'); /* failed checksum */
– else {
– putDebugChar('+'); /* successful transfer */
– /* if a sequence char is present, reply the sequence ID */
– if (buffer[2] == ':') {
– putDebugChar(buffer[0]);
– putDebugChar(buffer[1]);
– /* remove sequence chars from buffer */
– count = strlen(buffer);
– for (i = 3; i <= count; i++)
– buffer[i - 3] = buffer[i];
– }
– }
– } while (checksum != xmitcsum);
–
– if (remote_debug)
– printk("R:%s\n", buffer);
– flushDebugChar();
–}

```

```

-
-/* send the packet in buffer. */
-
-void
-putpacket(char *buffer)
- {
-     unsigned char checksum;
-     int count;
-     char ch;
-
-     /* $<packet info>#<checksum>. */
-
-     if (kgdb_eth == -1) {
-     do {
-     if (remote_debug)
-     printk("T:%s\n", buffer);
-     putDebugChar('$');
-     checksum = 0;
-     count = 0;
-
-     while ((ch = buffer[count])) {
-     putDebugChar(ch);
-     checksum += ch;
-     count += 1;
-     }
-
-     putDebugChar('#');
-     putDebugChar(hexchars[checksum >> 4]);
-     putDebugChar(hexchars[checksum % 16]);
-     flushDebugChar();
-
-     } while ((getDebugChar() & 0x7f) != '+');
-     } else {
-     /*
-     * For udp, we can not transfer too much bytes once.
-     * We only transfer MAX_SEND_COUNT size bytes each time
-     */
-
-     #define MAX_SEND_COUNT 30
-
-     int send_count = 0, i = 0;
-     char send_buf[MAX_SEND_COUNT];
-
-     do {
-     if (remote_debug)
-     printk("T:%s\n", buffer);
-     putDebugChar('$');
-     checksum = 0;
-     count = 0;
-     send_count = 0;
-     while ((ch = buffer[count])) {

```

Linux–Kernel: kgdb cleanups

```
– if (send_count >= MAX_SEND_COUNT) {
– for(i = 0; i < MAX_SEND_COUNT; i++) {
– putDebugChar(send_buf[i]);
– }
– flushDebugChar();
– send_count = 0;
– } else {
– send_buf[send_count] = ch;
– checksum += ch;
– count ++;
– send_count++;
– }
– }
– for(i = 0; i < send_count; i++)
– putDebugChar(send_buf[i]);
– putDebugChar('#');
– putDebugChar(hexchars[checksum >> 4]);
– putDebugChar(hexchars[checksum % 16]);
– flushDebugChar();
– } while ((getDebugChar() & 0x7f) != '+');
– }
– }
–
–static char remcomInBuffer[BUFMAX];
–static char remcomOutBuffer[BUFMAX];
–static short error;
–
–void
–debug_error(char *format, char *parm)
–{
– if (remote_debug)
– printk(format, parm);
– }

static void
print_regs(struct pt_regs *regs)
@@ -658,14 +215,10 @@
#endif

} /* gdb_regs_to_regs */
–extern void scheduling_functions_start_here(void);
–extern void scheduling_functions_end_here(void);
–#define first_sched ((unsigned long) scheduling_functions_start_here)
–#define last_sched ((unsigned long) scheduling_functions_end_here)

int thread_list = 0;

–void
+static void
get_gdb_regs(struct task_struct *p, struct pt_regs *regs, int *gdb_regs)
{
```

```

    unsigned long stack_page;
@@ -727,20 +280,7 @@
    return;
}

-/* Indicate to caller of mem2hex or hex2mem that there has been an
- error. */
-static volatile int mem_err = 0;
-static volatile int mem_err_expected = 0;
-static volatile int mem_err_cnt = 0;
-static int garbage_loc = -1;
-
-int
-get_char(char *addr)
-{
- return *addr;
-}
-
-void
+static void
set_char(char *addr, int val, int may_fault)
{
    /*
@@ -756,260 +296,7 @@
    *addr = val;
}

-/* convert the memory pointed to by mem into hex, placing result in buf */
-/* return a pointer to the last char put in buf (null) */
-/* If MAY_FAULT is non-zero, then we should set mem_err in response to
- a fault; if zero treat a fault like any other fault in the stub. */
-char *
-mem2hex(char *mem, char *buf, int count, int may_fault)
-{
- int i;
- unsigned char ch;
-
- if (may_fault) {
- mem_err_expected = 1;
- mem_err = 0;
- }
- for (i = 0; i < count; i++) {
- /* printk("%lx = ", mem); */
-
- ch = get_char(mem++);
-
- /* printk("%02x\n", ch & 0xFF); */
- if (may_fault && mem_err) {
- if (remote_debug)
- printk("Mem fault fetching from addr %lx\n",
- (long) (mem - 1));

```

Linux–Kernel: kgdb cleanups

```
– *buf = 0; /* truncate buffer */
– return (buf);
– }
– *buf++ = hexchars[ch >> 4];
– *buf++ = hexchars[ch % 16];
– }
– *buf = 0;
– if (may_fault)
– mem_err_expected = 0;
– return (buf);
– }
–
–/* convert the hex array pointed to by buf into binary to be placed in mem */
–/* return a pointer to the character AFTER the last byte written */
–/* NOTE: We use the may fault flag to also indicate if the write is to
– * the registers (0) or "other" memory (!=0)
– */
–char *
–hex2mem(char *buf, char *mem, int count, int may_fault)
–{
– int i;
– unsigned char ch;
–
– if (may_fault) {
– mem_err_expected = 1;
– mem_err = 0;
– }
– for (i = 0; i < count; i++) {
– ch = hex(*buf++) << 4;
– ch = ch + hex(*buf++);
– set_char(mem++, ch, may_fault);
–
– if (may_fault && mem_err) {
– if (remote_debug)
– printk("Mem fault storing to addr %lx\n",
– (long) (mem – 1));
– return (mem);
– }
– }
– if (may_fault)
– mem_err_expected = 0;
– return (mem);
– }
–
–/*****
–/* WHILE WE FIND NICE HEX CHARS, BUILD AN INT */
–/* RETURN NUMBER OF CHARS PROCESSED */
–/*****
–int
–hexToInt(char **ptr, int *intValue)
–{
```

```

- int numChars = 0;
- int hexValue;

- *intValue = 0;
-
- while (**ptr) {
- hexValue = hex(**ptr);
- if (hexValue >= 0) {
- *intValue = (*intValue << 4) | hexValue;
- numChars++;
- } else
- break;
-
- (*ptr)++;
- }
-
- return (numChars);
-}
-
-#define stubhex(h) hex(h)
-#ifdef old_thread_list
-
- static int
- stub_unpack_int(char *buff, int fieldlength)
- {
- int nibble;
- int retval = 0;
-
- while (fieldlength) {
- nibble = stubhex(*buff++);
- retval |= nibble;
- fieldlength--;
- if (fieldlength)
- retval = retval << 4;
- }
- return retval;
- }
-#endif
- static char *
- pack_hex_byte(char *pkt, int byte)
- {
- *pkt++ = hexchars[(byte >> 4) & 0xf];
- *pkt++ = hexchars[(byte & 0xf)];
- return pkt;
- }
-
-#define BUF_THREAD_ID_SIZE 16
-
- static char *
- pack_threadid(char *pkt, threadref * id)
- {

```

```

- char *limit;
- unsigned char *altid;
-
- altid = (unsigned char *) id;
- limit = pkt + BUF_THREAD_ID_SIZE;
- while (pkt < limit)
- pkt = pack_hex_byte(pkt, *altid++);
- return pkt;
-}
-
-#ifdef old_thread_list
-static char *
-unpack_byte(char *buf, int *value)
-{
- *value = stub_unpack_int(buf, 2);
- return buf + 2;
-}
-
-static char *
-unpack_threadid(char *inbuf, threadref * id)
-{
- char *altref;
- char *limit = inbuf + BUF_THREAD_ID_SIZE;
- int x, y;
-
- altref = (char *) id;
-
- while (inbuf < limit) {
- x = stubhex(*inbuf++);
- y = stubhex(*inbuf++);
- *altref++ = (x << 4) | y;
- }
- return inbuf;
-}
-#endif
-void
-int_to_threadref(threadref * id, int value)
-{
- unsigned char *scan;
-
- scan = (unsigned char *) id;
- {
- int i = 4;
- while (i--)
- *scan++ = 0;
- }
- *scan++ = (value >> 24) & 0xff;
- *scan++ = (value >> 16) & 0xff;
- *scan++ = (value >> 8) & 0xff;
- *scan++ = (value & 0xff);
-}

```

```

-int
-int_to_hex_v(unsigned char * id, int value)
-{
- unsigned char *start = id;
- int shift;
- int ch;
-
-
- for (shift = 28; shift >= 0; shift -= 4) {
- if ((ch = (value >> shift) & 0xf) || (id != start)) {
- *id = hexchars[ch];
- id++;
- }
- }
- if (id == start)
- *id++ = '0';
- return id - start;
-}
-#ifdef old_thread_list
-
-
-#static int
-#threadref_to_int(threadref * ref)
-#{
-# int i, value = 0;
-# unsigned char *scan;
-#
-# scan = (char *) ref;
-# scan += 4;
-# i = 4;
-# while (i-- > 0)
-# value = (value << 8) | ((*scan++) & 0xff);
-# return value;
-#}
-#endif
-#static int
-#cmp_str(char *s1, char *s2, int count)
-#{
-# while (count--) {
-# if (*s1++ != *s2++)
-# return 0;
-# }
-# return 1;
-#}
-
-
-#if 1 /* this is a hold over from 2.4 where O(1) was "sometimes" */
-#extern struct task_struct *kgdb_get_idle(int cpu);
-#define idle_task(cpu) kgdb_get_idle(cpu)
-#else
-#define idle_task(cpu) init_tasks[cpu]
-#endif
-
-
-#extern int kgdb_pid_init_done;

```

```

-
-struct task_struct *
-getthread(int pid)
- {
- struct task_struct *thread;
- if (pid >= PID_MAX && pid <= (PID_MAX + MAX_NO_CPUS)) {
-
- return idle_task(pid - PID_MAX);
- } else {
- /*
- * find_task_by_pid is relatively safe all the time
- * Other pid functions require lock downs which imply
- * that we may be interrupting them (as we get here
- * in the middle of most any lock down).
- * Still we don't want to call until the table exists!
- */
- if (kgdb_pid_init_done){
- thread = find_task_by_pid(pid);
- if (thread) {
- return thread;
- }
- }
- }
- return NULL;
- }
-/* *INDENT-OFF* */
struct hw_breakpoint {
    unsigned enabled;
    unsigned type;
@@ -1019,7 +306,7 @@
    {enabled:0},
    {enabled:0},
    {enabled:0}};
-/* *INDENT-ON* */
+
unsigned hw_breakpoint_status;
void
correct_hw_break(void)
@@ -1031,7 +318,7 @@

    asm volatile ("movl %%db7, %0\n": "=r" (dr7)
                  :);
-/* *INDENT-OFF* */
+
do {
    unsigned addr0, addr1, addr2, addr3;
    asm volatile ("movl %%db0, %0\n"
@@ -1042,7 +329,7 @@
                  "=r"(addr2), "=r"(addr3)
                  :);
    } while (0);

```

Linux–Kernel: kgdb cleanups

```
– /* *INDENT–ON* */
+
    correctit = 0;
    for (breakno = 0; breakno < 3; breakno++) {
        breakbit = 2 << (breakno << 1);
@@ –1095,7 +382,7 @@
        return 0;
    }

–int
+static int
set_hw_break(unsigned breakno, unsigned type, unsigned len, unsigned addr)
{
    if (breakinfo[breakno].enabled) {
@@ –1174,9 +461,6 @@
        in_kgdb_here_log[cpu] = 0;
        kgdb_local_irq_restore(flags);
        return 1;
– /*
– spin_unlock(continuelocks + smp_processor_id());
– */
    }

void
@@ –1298,16 +582,6 @@
    * released.
    */
#ifdef CONFIG_SMP
–
–#if 0
– if (cpu_callout_map & ~MAX_CPU_MASK) {
– printk("kgdb : too many cpus, possibly not mapped"
– " in contiguous space, change MAX_NO_CPUS"
– " in kgdb_stub and make new kernel.\n"
– " cpu_callout_map is %lx\n", cpu_callout_map);
– goto exit_just_unlock;
– }
–#endif
    if (spinlock_count == 1) {
        int time, end_time, dum;
        int i;
@@ –1388,25 +662,7 @@
        if (i < num_online_cpus()) {
            printk
                ("kgdb : time out, proceeding without sync\n");
–#if 0
– printk("kgdb : Waiting_cpus: 0 = %d, 1 = %d\n",
– waiting_cpus[0].task != 0,
– waiting_cpus[1].task != 0);
– printk("kgdb : Cpu_logged in: 0 = %d, 1 = %d\n",
– cpu_logged_in[0], cpu_logged_in[1]);
```

Linux–Kernel: kgdb cleanups

```
- printk
- ("kgdb : in_kgdb_here_log in: 0 = %d, 1 = %d\n",
- in_kgdb_here_log[0] != 0,
- in_kgdb_here_log[1] != 0);
-#endif
                entry_state = NO_SYNC;
- } else {
-#if 0
- int ent =
- in_kgdb_entry_log[smp_processor_id()] -
- me_in_kgdb;
- printk("kgdb : sync after %d entries\n", ent);
-#endif
        }
        } else {
            if (remote_debug) {
@@ -1434,7 +690,7 @@

                /* Disable hardware debugging while we are in kgdb */
                /* Get the debug register status register */
-/* *INDENT-OFF* */
+
        __asm__("movl %0,%%db7"
                : /* no output */
                : "r"(0));
@@ -1443,7 +699,6 @@
                : "r" (hw_breakpoint_status)
                :);

-/* *INDENT-ON* */
        switch (exceptionVector) {
        case 0: /* divide error */
        case 1: /* debug exception */
@@ -1533,7 +788,7 @@
        }

        kgdb_eth_reply_arp();
- while (1 == 1) {
+ while (1) {
            error = 0;
            remcomOutBuffer[0] = 0;
            getpacket(remcomInBuffer);
@@ -1989,11 +1244,8 @@

                /* reply to the request */
                putpacket(remcomOutBuffer);
- } /* while(1==1) */
- /*
- * reached by goto only.
- */
- exit_kgdb:
```

Linux–Kernel: kgdb cleanups

```
+ } /* while(1) */
+exit_kgdb: /* reached by goto only. */
/*
 * Here is where we set up to trap a gdb function call. NEW_esp
 * will be changed if we are trying to do this. We handle both
@@ -2072,9 +1324,6 @@
    kgdb_local_irq_restore(flags);
    return (0);
}
-#if 0
-#endif
-#endif
/* Release kgdb spinlock */
KGDB_SPIN_UNLOCK(&kgdb_spinlock);
@@ -2106,7 +1355,7 @@

/* In case GDB is started before us, ack any packets (presumably
"$?#xx") sitting there.
- putDebugChar ('+');
+ put_debug_char ('+');

    initialized = 1;
    */
@@ -2119,27 +1368,7 @@
/* But really, just use the BREAKPOINT macro. We will handle the int stuff
*/

-#ifdef later
-/*
- * possibly we should not go thru the traps.c code at all? Someday.
- */
-void
-do_kgdb_int3(struct pt_regs *regs, long error_code)
-{
- kgdb_handle_exception(3, 5, error_code, regs);
- return;
-}
-#endif
#undef regs
-#ifdef CONFIG_TRAP_BAD_SYSCALL_EXITS
-asmlinkage void
-bad_sys_call_exit(int stuff)
-{
- struct pt_regs *regs = (struct pt_regs *) &stuff;
- printk("Sys call %d return with %x preempt_count\n",
- (int) regs->orig_eax, preempt_count());
-}
-#endif
#ifdef CONFIG_STACK_OVERFLOW_TEST
#include <asm/kgdb.h>
```

Linux–Kernel: kgdb cleanups

```
asmlinkage void
@@ -2150,8 +1379,7 @@
#else
    printk("Kernel stack overflow, looping forever\n");
#endif
- while (1) {
- }
+ while (1) ;
}
#endif

@@ -2420,9 +1648,6 @@
    kgdb_and_then = &kgdb_data[++kgdb_and_then_count & INDEX_MASK];
    spin_unlock(&ts_spin);
    kgdb_local_irq_restore(flags);
-#ifdef CONFIG_PREEMPT
-
-#endif
    return;
}
#endif
@@ -2483,7 +1708,6 @@
    return parse_hw_addr(str, kgdb_localmac);
}

-
__setup("gdbeth=", kgdb_opt_kgdbeth);
__setup("gdbeth_remoteip=", kgdb_opt_kgdbeth_remoteip);
__setup("gdbeth_listenport=", kgdb_opt_kgdbeth_listenport);
--- tmp/linux/drivers/net/kgdb_eth.c 2004-01-09 19:04:43.000000000 +0100
+++ linux/drivers/net/kgdb_eth.c 2003-12-28 21:00:24.000000000 +0100
@@ -39,12 +39,12 @@

#define GDB_BUF_SIZE 512 /* power of 2, please */

-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 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;

-extern void set_debug_traps(void) ; /* GDB routine */
+extern void set_debug_traps(void); /* GDB routine */
extern void breakpoint(void);

unsigned int kgdb_remoteip = 0;
@@ -479,8 +479,7 @@
put_char_on_queue(int chr)
```

Linux-Kernel: kgdb cleanups

```
{
    eth_queue[outgoing_queue++] = chr;
- if(outgoing_queue == ETH_QUEUE_SIZE)
- {
+ if(outgoing_queue == ETH_QUEUE_SIZE) {
    eth_flushDebugChar();
    }
}
--- tmp/linux/include/asm-i386/kgdb_local.h 2004-01-09 19:04:44.000000000 +0100
+++ linux/include/asm-i386/kgdb_local.h 2003-12-28 21:21:55.000000000 +0100
@@ -83,7 +83,7 @@
    tty: (struct tty_struct *)&state, \
    IER: SB_IER, \
    MCR: SB_MCR}
-extern void putDebugChar(int);
+extern void put_debug_char(int);
/* RTAI support needs us to really stop/start interrupts */

#define kgdb_sti() __asm__ __volatile__("sti": : : "memory")
@@ -98,5 +98,5 @@
#ifdef CONFIG_SERIAL
extern void shutdown_for_kgdb(struct async_struct *info);
#endif
-#define INIT_KDEBUG putDebugChar("+");
+#define INIT_KDEBUG put_debug_char("+"); /* WTF is this? it expects integer! */
#endif /* __KGDB_LOCAL */
--- tmp/linux/include/linux/dwarf2-lang.h 2004-01-09 19:04:44.000000000 +0100
+++ linux/include/linux/dwarf2-lang.h 2003-12-27 17:12:38.000000000 +0100
@@ -1,132 +0,0 @@
-#ifndef DWARF2_LANG
-#define DWARF2_LANG
-#include <linux/dwarf2.h>
-
-/*
- * This 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, or (at your option) any later
- * version.
- */
-/*
- * This file defines macros that allow generation of DWARF debug records
- * for asm files. This file is platform independent. Register numbers
- * (which are about the only thing that is platform dependent) are to be
- * supplied by a platform defined file.
- */
-#define DWARF_preamble() .section .debug_frame,"",@progbits
-/*
- * This macro starts a debug frame section. The debug_frame describes
- * where to find the registers that the enclosing function saved on
- * entry.
- */
-*
```

Linux–Kernel: kgdb cleanups

```
– * ORD is use by the label generator and should be the same as what is
– * passed to CFI_postamble.
– *
– * pc, pc register gdb ordinal.
– *
– * code_align this is the factor used to define locations or regions
– * where the given definitions apply. If you use labels to define these
– * this should be 1.
– *
– * data_align this is the factor used to define register offsets. If
– * you use struct offset, this should be the size of the register in
– * bytes or the negative of that. This is how it is used: you will
– * define a register as the reference register, say the stack pointer,
– * then you will say where a register is located relative to this
– * reference registers value, say 40 for register 3 (the gdb register
– * number). The <40> will be multiplied by <data_align> to define the
– * byte offset of the given register (3, in this example). So if your
– * <40> is the byte offset and the reference register points at the
– * begining, you would want 1 for the data_offset. If <40> was the 40th
– * 4–byte element in that structure you would want 4. And if your
– * reference register points at the end of the structure you would want
– * a negative data_align value(and you would have to do other math as
– * well).
– */
–
–#define CFI_preamble(ORD, pc, code_align, data_align) \
–.section .debug_frame,"",@progbits ; \
–frame/**/_**/ORD: \
–.long end/**/_**/ORD–start/**/_**/ORD; \
–start/**/_**/ORD: \
–.long DW_CIE_ID; \
–.byte DW_CIE_VERSION; \
–.byte 0 ; \
–.uleb128 code_align; \
–.sleb128 data_align; \
–.byte pc;
–
–/*
– * After the above macro and prior to the CFI_postamble, you need to
– * define the initial state. This starts with defining the reference
– * register and, usually the pc. Here are some helper macros:
– */
–
–#define CFA_define_reference(reg, offset) \
–.byte DW_CFA_def_cfa; \
–.uleb128 reg; \
–.uleb128 (offset);
–
–#define CFA_define_offset(reg, offset) \
–.byte (DW_CFA_offset + reg); \
–.uleb128 (offset);
```

```

-
-#define CFI_postamble(ORD) \
- .align 4; \
- end/**/_/**/ORD:
-/*
- * So now your code pushes stuff on the stack, you need a new location
- * and the rules for what to do. This starts a running description of
- * the call frame. You need to describe what changes with respect to
- * the call registers as the location of the pc moves through the code.
- * The following builds an FDE (fram descriptor entry?). Like the
- * above, it has a preamble and a postamble. It also is tied to the CFI
- * above.
- * The first entry after the preamble must be the location in the code
- * that the call frame is being described for.
- */
-#define FDE_preamble(ORD, fde_no, initial_address, length) \
- .long FDE_end/**/_**/fde_no-FDE_start/**/_**/fde_no; \
- FDE_start/**/_**/fde_no: \
- .long frame/**/_**/ORD; \
- .long initial_address; \
- .long length;
-
-#define FDE_postamble(fde_no) \
- .align 4; \
- FDE_end/**/_**/fde_no:
-/*
- * That done, you can now add registers, subtract registers, move the
- * reference and even change the reference. You can also define a new
- * area of code the info applies to. For discontinuous bits you should
- * start a new FDE. You may have as many as you like.
- */
-
-/*
- * To advance the address by <bytes>
- */
-
-#define FDE_advance(bytes) \
- .byte DW_CFA_advance_loc4 \
- .long bytes
-
-
-
-/*
- * With the above you can define all the register locations. But
- * suppose the reference register moves... Takes the new offset NOT an
- * increment. This is how esp is tracked if it is not saved.
- */
-
-#define CFA_define_cfa_offset(offset) \
- .byte $DW_CFA_def_cfa_offset; \
- .uleb128 (offset);

```

Linux–Kernel: kgdb cleanups

```

_/*
- * Or suppose you want to use a different reference register...
- */
-#define CFA_define_cfa_register(reg) \
- .byte DW_CFA_def_cfa_register; \
- .uleb128 reg;
-
-#endif
--- tmp/linux/include/linux/dwarf2.h 2004-01-09 19:04:44.000000000 +0100
+++ linux/include/linux/dwarf2.h 2003-12-27 17:12:40.000000000 +0100
@@ -1,738 +0,0 @@
_/* Declarations and definitions of codes relating to the DWARF2 symbolic
- debugging information format.
- Copyright (C) 1992, 1993, 1995, 1996, 1997, 1999, 2000, 2001, 2002
- Free Software Foundation, Inc.
-
- Written by Gary Funck (gary@intrepid.com) The Ada Joint Program
- Office (AJPO), Florida State Unviversity and Silicon Graphics Inc.
- provided support for this effort -- June 21, 1995.
-
- Derived from the DWARF 1 implementation written by Ron Guilmette
- (rfg@netcom.com), November 1990.
-
- This file is part of GCC.
-
- GCC 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, or (at your option) any later
- version.
-
- GCC is distributed in the hope that it will be useful, but WITHOUT
- ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
- or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public
- License for more details.
-
- You should have received a copy of the GNU General Public License
- along with GCC; see the file COPYING. If not, write to the Free
- Software Foundation, 59 Temple Place – Suite 330, Boston, MA
- 02111-1307, USA. */
-
_/* This file is derived from the DWARF specification (a public document)
- Revision 2.0.0 (July 27, 1993) developed by the UNIX International
- Programming Languages Special Interest Group (UI/PLSIG) and distributed
- by UNIX International. Copies of this specification are available from
- UNIX International, 20 Waterview Boulevard, Parsippany, NJ, 07054.
-
- This file also now contains definitions from the DWARF 3 specification. */
-
_/* This file is shared between GCC and GDB, and should not contain
- prototypes. */
-
```


Linux–Kernel: kgdb cleanups

```
–typedef struct  
–{  
– unsigned char cu_length [4];  
– unsigned char cu_version [2];  
– unsigned char cu_abbrev_offset [4];  
– unsigned char cu_p
```