

[PATCH,RFC 2.6.14 08/15] KGDB: x86_64-specific changes

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-11/3761.html>

From: Tom Rini (trini_at_kernel.crashing.org)

Date: 11/10/05

To: Andrew Morton <akpm@osdl.org>

Date: Thu, 10 Nov 2005 11:41:49 -0500

This adds support for the x86_64 architecture. In addition to what was noted in the core-lite patch about stuff outside of new files, we add -g0 to compiling of syscalls.o as otherwise we run into problems when debugging modules, and like i386 annotate switch_to().

```
arch/x86_64/Kconfig.debug | 3
arch/x86_64/kernel/Makefile | 1
arch/x86_64/kernel/kgdb-jmp.S | 65 ++++++
arch/x86_64/kernel/kgdb.c | 460 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
include/asm-x86_64/kgdb.h | 50 +++++
include/asm-x86_64/system.h | 6
include/linux/kgdb.h | 4
kernel/kgdb.c | 14 +
lib/Kconfig.debug | 2
9 files changed, 599 insertions(+), 6 deletions(-)
```

Index: linux-2.6.14/arch/x86_64/Kconfig.debug

```
-----
--- linux-2.6.14.orig/arch/x86_64/Kconfig.debug
+++ linux-2.6.14/arch/x86_64/Kconfig.debug
@@ -51,7 +51,4 @@ config IOMMU_LEAK
     Add a simple leak tracer to the IOMMU code. This is useful when you
     are debugging a buggy device driver that leaks IOMMU mappings.
```

```
---#config X86_REMOTE_DEBUG
---# bool "kgdb debugging stub"
```

```
---
---endmenu
```

Index: linux-2.6.14/arch/x86_64/kernel/kgdb.c

```
-----
--- /dev/null
+++ linux-2.6.14/arch/x86_64/kernel/kgdb.c
@@ -0,0 +1,460 @@
+/*
+ *
```

Linux–Kernel: [PATCH,RFC 2.6.14 08/15] KGDB: x86_64–specific changes

```
+ * 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, or (at your option) any
+ * later version.
+ *
+ * This program 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.
+ *
+ */
+
+/*
+ * Copyright (C) 2004 Amit S. Kale <amitkale@linsyssoft.com>
+ * Copyright (C) 2000–2001 VERITAS Software Corporation.
+ * Copyright (C) 2002 Andi Kleen, SuSE Labs
+ * Copyright (C) 2004 LinSysSoft Technologies Pvt. Ltd.
+ */
+/******
+ * Contributor: Lake Stevens Instrument Division$
+ * Written by: Glenn Engel $
+ * Updated by: Amit Kale<akale@veritas.com>
+ * Modified for 386 by Jim Kingdon, Cygnus Support.
+ * Orignal kgdb, compatibility with 2.1.xx kernel by
+ * David Grothe <dave@gcom.com>
+ * Integrated into 2.2.5 kernel by Tigran Aivazian <tigran@sco.com>
+ * X86_64 changes from Andi Kleen's patch merged by Jim Houston
+ */
+
+#include <linux/string.h>
+#include <linux/kernel.h>
+#include <linux/sched.h>
+#include <linux/smp.h>
+#include <linux/spinlock.h>
+#include <linux/delay.h>
+#include <asm/system.h>
+#include <asm/ptrace.h> /* for linux pt_regs struct */
+#include <linux/kgdb.h>
+#include <linux/init.h>
+#include <asm/apicdef.h>
+#include <asm/mach_apic.h>
+#include <asm/kdebug.h>
+#include <asm/debugreg.h>
+
+/* Put the error code here just in case the user cares. */
+int gdb_x86_64errcode;
+/* Likewise, the vector number here (since GDB only gets the signal
+ number through the usual means, and that's not very specific). */
+int gdb_x86_64vector = -1;
+
+extern atomic_t cpu_doing_single_step;
```

```

+
+void regs_to_gdb_regs(unsigned long *gdb_regs, struct pt_regs *regs)
+{
+ gdb_regs[_RAX] = regs->rax;
+ gdb_regs[_RBX] = regs->rbx;
+ gdb_regs[_RCX] = regs->rcx;
+ gdb_regs[_RDX] = regs->rdx;
+ gdb_regs[_RSI] = regs->rsi;
+ gdb_regs[_RDI] = regs->rdi;
+ gdb_regs[_RBP] = regs->rbp;
+ gdb_regs[_PS] = regs->eflags;
+ gdb_regs[_PC] = regs->rip;
+ gdb_regs[_R8] = regs->r8;
+ gdb_regs[_R9] = regs->r9;
+ gdb_regs[_R10] = regs->r10;
+ gdb_regs[_R11] = regs->r11;
+ gdb_regs[_R12] = regs->r12;
+ gdb_regs[_R13] = regs->r13;
+ gdb_regs[_R14] = regs->r14;
+ gdb_regs[_R15] = regs->r15;
+ gdb_regs[_RSP] = regs->rsp;
+}
+
+extern void thread_return(void);
+void sleeping_thread_to_gdb_regs(unsigned long *gdb_regs, struct task_struct *p)
+{
+ gdb_regs[_RAX] = 0;
+ gdb_regs[_RBX] = 0;
+ gdb_regs[_RCX] = 0;
+ gdb_regs[_RDX] = 0;
+ gdb_regs[_RSI] = 0;
+ gdb_regs[_RDI] = 0;
+ gdb_regs[_RBP] = *(unsigned long *)p->thread.rsp;
+ gdb_regs[_PS] = *(unsigned long *)p->thread.rsp + 8;
+ gdb_regs[_PC] = (unsigned long)&thread_return;
+ gdb_regs[_R8] = 0;
+ gdb_regs[_R9] = 0;
+ gdb_regs[_R10] = 0;
+ gdb_regs[_R11] = 0;
+ gdb_regs[_R12] = 0;
+ gdb_regs[_R13] = 0;
+ gdb_regs[_R14] = 0;
+ gdb_regs[_R15] = 0;
+ gdb_regs[_RSP] = p->thread.rsp;
+}
+
+void gdb_regs_to_regs(unsigned long *gdb_regs, struct pt_regs *regs)
+{
+ regs->rax = gdb_regs[_RAX];
+ regs->rbx = gdb_regs[_RBX];
+ regs->rcx = gdb_regs[_RCX];

```

```

+ regs->rdx = gdb_regs[_RDX];
+ regs->rsi = gdb_regs[_RSI];
+ regs->rdi = gdb_regs[_RDI];
+ regs->rbp = gdb_regs[_RBP];
+ regs->eflags = gdb_regs[_PS];
+ regs->rip = gdb_regs[_PC];
+ regs->r8 = gdb_regs[_R8];
+ regs->r9 = gdb_regs[_R9];
+ regs->r10 = gdb_regs[_R10];
+ regs->r11 = gdb_regs[_R11];
+ regs->r12 = gdb_regs[_R12];
+ regs->r13 = gdb_regs[_R13];
+ regs->r14 = gdb_regs[_R14];
+ regs->r15 = gdb_regs[_R15];
+ #if 0 /* can't change these */
+ regs->rsp = gdb_regs[_RSP];
+ regs->ss = gdb_regs[_SS];
+ regs->fs = gdb_regs[_FS];
+ regs->gs = gdb_regs[_GS];
+ #endif
+
+ } /* gdb_regs_to_regs */
+
+ struct hw_breakpoint {
+   unsigned enabled;
+   unsigned type;
+   unsigned len;
+   unsigned long addr;
+ } breakinfo[4] = { {
+   +enabled:0}, {
+   +enabled:0}, {
+   +enabled:0}, {
+   +enabled:0} };
+
+ void kgdb_correct_hw_break(void)
+ {
+   int breakno;
+   int correctit;
+   int breakbit;
+   unsigned long dr7;
+
+   asm volatile ("movq %%db7, %0\n":"=r" (dr7):);
+   do {
+     unsigned long addr0, addr1, addr2, addr3;
+     asm volatile ("movq %%db0, %0\n"
+ "movq %%db1, %1\n"
+ "movq %%db2, %2\n"
+ "movq %%db3, %3\n":"=r" (addr0), "=r"(addr1),
+ "=r"(addr2), "=r"(addr3):);
+   } while (0);
+   correctit = 0;

```

```

+ for (breakno = 0; breakno < 3; breakno++) {
+ breakbit = 2 << (breakno << 1);
+ if (!(dr7 & breakbit) && breakinfo[breakno].enabled) {
+ correctit = 1;
+ dr7 |= breakbit;
+ dr7 &= ~(0xf0000 << (breakno << 2));
+ dr7 |= (((breakinfo[breakno].len << 2) |
+ breakinfo[breakno].type) << 16) <<
+ (breakno << 2);
+ switch (breakno) {
+ case 0:
+ asm volatile ("movq %0, %%dr0\n":"r"
+ (breakinfo[breakno].addr));
+ break;
+
+ case 1:
+ asm volatile ("movq %0, %%dr1\n":"r"
+ (breakinfo[breakno].addr));
+ break;
+
+ case 2:
+ asm volatile ("movq %0, %%dr2\n":"r"
+ (breakinfo[breakno].addr));
+ break;
+
+ case 3:
+ asm volatile ("movq %0, %%dr3\n":"r"
+ (breakinfo[breakno].addr));
+ break;
+ }
+ } else if ((dr7 & breakbit) && !breakinfo[breakno].enabled) {
+ correctit = 1;
+ dr7 &= ~breakbit;
+ dr7 &= ~(0xf0000 << (breakno << 2));
+ }
+ }
+ if (correctit) {
+ asm volatile ("movq %0, %%db7\n":"r" (dr7));
+ }
+ }
+
+int kgdb_remove_hw_break(unsigned long addr)
+{
+ int i, idx = -1;
+ for (i = 0; i < 4; i++) {
+ if (breakinfo[i].addr == addr && breakinfo[i].enabled) {
+ idx = i;
+ break;
+ }
+ }
+ if (idx == -1)

```

```

+ return -1;
+
+ breakinfo[idx].enabled = 0;
+ return 0;
+}
+
+int kgdb_set_hw_break(unsigned long addr)
+{
+ int i, idx = -1;
+ for (i = 0; i < 4; i++) {
+ if (!breakinfo[i].enabled) {
+ idx = i;
+ break;
+ }
+ }
+ if (idx == -1)
+ return -1;
+
+ breakinfo[idx].enabled = 1;
+ breakinfo[idx].type = 1;
+ breakinfo[idx].len = 1;
+ breakinfo[idx].addr = addr;
+ return 0;
+}
+
+int remove_hw_break(unsigned breakno)
+{
+ if (!breakinfo[breakno].enabled) {
+ return -1;
+ }
+ breakinfo[breakno].enabled = 0;
+ return 0;
+}
+
+int set_hw_break(unsigned breakno, unsigned type, unsigned len, unsigned addr)
+{
+ if (breakinfo[breakno].enabled) {
+ return -1;
+ }
+ breakinfo[breakno].enabled = 1;
+ breakinfo[breakno].type = type;
+ breakinfo[breakno].len = len;
+ breakinfo[breakno].addr = addr;
+ return 0;
+}
+
+void kgdb_disable_hw_debug(struct pt_regs *regs)
+{
+ /* Disable hardware debugging while we are in kgdb */
+ asm volatile ("movq %0,%%db7": /* no output */ : "r" (OUL));
+}

```

```

+
+void kgdb_post_master_code(struct pt_regs *regs, int e_vector, int err_code)
+{
+ /* Master processor is completely in the debugger */
+ gdb_x86_64vector = e_vector;
+ gdb_x86_64errcode = err_code;
+}
+
+void kgdb_roundup_cpus(unsigned long flags)
+{
+ send_IPI_allbutself(APIC_DM_NMI);
+}
+
+int kgdb_arch_handle_exception(int e_vector, int signo, int err_code,
+ char *remcomInBuffer, char *remcomOutBuffer,
+ struct pt_regs *linux_regs)
+{
+ unsigned long addr, length;
+ unsigned long breakno, breaktype;
+ char *ptr;
+ int newPC;
+ unsigned long dr6;
+
+ switch (remcomInBuffer[0]) {
+ case 'c':
+ case 's':
+ /* try to read optional parameter, pc unchanged if no parm */
+ ptr = &remcomInBuffer[1];
+ if (kgdb_hex2long(&ptr, &addr))
+ linux_regs->rip = addr;
+ newPC = linux_regs->rip;
+
+ /* clear the trace bit */
+ linux_regs->eflags &= ~TF_MASK;
+
+ atomic_set(&cpu_doing_single_step, -1);
+ /* set the trace bit if we're stepping */
+ if (remcomInBuffer[0] == 's') {
+ linux_regs->eflags |= TF_MASK;
+ debugger_step = 1;
+ if (kgdb_contthread)
+ atomic_set(&cpu_doing_single_step,
+ smp_processor_id());
+ }
+
+ asm volatile ("movq %%db6, %0\n":"=r" (dr6));
+ if (!(dr6 & 0x4000)) {
+ for (breakno = 0; breakno < 4; ++breakno) {
+ if (dr6 & (1 << breakno)) {
+ if (breakinfo[breakno].type == 0) {

```

```

+ /* Set restore flag */
+ linux_regs->eflags |=
+ X86_EFLAGS_RF;
+ break;
+ }
+ }
+ }
+ }
+ kgdb_correct_hw_break();
+ asm volatile ("movq %0, %%db6\n"::"r" (OUL));
+
+ return (0);
+
+ case 'Y':
+ ptr = &remcomInBuffer[1];
+ kgdb_hex2long(&ptr, &breakno);
+ ptr++;
+ kgdb_hex2long(&ptr, &breaktype);
+ ptr++;
+ kgdb_hex2long(&ptr, &length);
+ ptr++;
+ kgdb_hex2long(&ptr, &addr);
+ if (set_hw_break(breakno & 0x3, breaktype & 0x3,
+ length & 0x3, addr) == 0)
+ strcpy(remcomOutBuffer, "OK");
+ else
+ strcpy(remcomOutBuffer, "ERROR");
+ break;
+
+ /* Remove hardware breakpoint */
+ case 'y':
+ ptr = &remcomInBuffer[1];
+ kgdb_hex2long(&ptr, &breakno);
+ if (remove_hw_break(breakno & 0x3) == 0)
+ strcpy(remcomOutBuffer, "OK");
+ else
+ strcpy(remcomOutBuffer, "ERROR");
+ break;
+
+ } /* switch */
+ return -1;
+ }
+
+static struct pt_regs *in_interrupt_stack(unsigned long rsp, int cpu)
+{
+ struct pt_regs *regs;
+ unsigned long end = (unsigned long)cpu_pda[cpu].irqstackptr;
+ if (rsp <= end && rsp >= end - IRQSTACKSIZE + 8) {
+ regs = *(((struct pt_regs **)end) - 1);
+ return regs;
+ }

```

```

+ return NULL;
+}
+
+static struct pt_regs *in_exception_stack(unsigned long rsp, int cpu)
+{
+ int i;
+ struct tss_struct *init_tss = &__get_cpu_var(init_tss);
+ for (i = 0; i < N_EXCEPTION_STACKS; i++)
+ if (rsp >= init_tss[cpu].ist[i] &&
+     rsp <= init_tss[cpu].ist[i] + EXCEPTION_STKSZ) {
+ struct pt_regs *r =
+ (void *)init_tss[cpu].ist[i] + EXCEPTION_STKSZ;
+ return r - 1;
+ }
+ return NULL;
+}
+
+void kgdb_shadowinfo(struct pt_regs *regs, char *buffer, unsigned threadid)
+{
+ static char intr_desc[] = "Stack at interrupt entrypoint";
+ static char exc_desc[] = "Stack at exception entrypoint";
+ struct pt_regs *stregs;
+ int cpu = hard_smp_processor_id();
+
+ if ((stregs = in_interrupt_stack(regs->rsp, cpu))
+     || kgdb_mem2hex(intr_desc, buffer, strlen(intr_desc));
+     || (stregs = in_exception_stack(regs->rsp, cpu))
+     || kgdb_mem2hex(exc_desc, buffer, strlen(exc_desc));
+ }
+
+struct task_struct *kgdb_get_shadow_thread(struct pt_regs *regs, int threadid)
+{
+ struct pt_regs *stregs;
+ int cpu = hard_smp_processor_id();
+
+ if ((stregs = in_interrupt_stack(regs->rsp, cpu))
+     || return current;
+     || (stregs = in_exception_stack(regs->rsp, cpu))
+     || return current;
+ }
+ return NULL;
+}
+
+struct pt_regs *kgdb_shadow_regs(struct pt_regs *regs, int threadid)
+{
+ struct pt_regs *stregs;
+ int cpu = hard_smp_processor_id();
+
+ if ((stregs = in_interrupt_stack(regs->rsp, cpu))
+     || return stregs;
+     || (stregs = in_exception_stack(regs->rsp, cpu))

```

```

+ return stregs;
+
+ return NULL;
+}
+
+/* Register KGDB with the die_chain so that we hook into all of the right
+ * spots. */
+static int kgdb_notify(struct notifier_block *self, unsigned long cmd,
+ void *ptr)
+{
+ struct die_args *args = ptr;
+ struct pt_regs *regs = args->regs;
+
+ if (cmd == DIE_PAGE_FAULT_NO_CONTEXT && atomic_read(&debugger_active)
+ && kgdb_may_fault) {
+ kgdb_fault_longjmp(kgdb_fault_jump_regs);
+ return NOTIFY_STOP;
+ /* CPU roundup? */
+ } else if (atomic_read(&debugger_active) && cmd == DIE_NMI_IPI) {
+ kgdb_nmihook(smp_processor_id(), regs);
+ return NOTIFY_STOP;
+ /* See if KGDB is interested. */
+ } else if (cmd == DIE_PAGE_FAULT || user_mode(regs) ||
+ cmd == DIE_NMI_IPI || (cmd == DIE_DEBUG &&
+ atomic_read(&debugger_active)))
+ /* Userpace events, normal watchdog event, or spurious
+ * debug exception. Ignore. */
+ return NOTIFY_DONE;
+
+ kgdb_handle_exception(args->trapnr, args->signr, args->err, regs);
+
+ return NOTIFY_STOP;
+}
+
+static struct notifier_block kgdb_notifier = {
+ .notifier_call = kgdb_notify,
+ .priority = 0x7fffffff, /* we need to notified first */
+};
+
+int kgdb_arch_init(void)
+{
+ notifier_chain_register(&die_chain, &kgdb_notifier);
+ return 0;
+}
+
+struct kgdb_arch arch_kgdb_ops = {
+ .gdb_bpt_instr = {0xcc},
+ .flags = KGDB_HW_BREAKPOINT,
+ .shadowth = 1,
+};

```

Index: linux-2.6.14/arch/x86_64/kernel/kgdb-jmp.S

```

----- /dev/null
+++ linux-2.6.14/arch/x86_64/kernel/kgdb-jmp.S
@@ -0,0 +1,65 @@
+/*
+ * arch/x86_64/kernel/kgdb-jmp.S
+ *
+ * Save and restore system registers so that within a limited frame we
+ * may have a fault and "jump back" to a known safe location.
+ *
+ * Author: Tom Rini <trini@kernel.crashing.org>
+ *
+ * Cribbed from glibc, which carries the following:
+ * Copyright (C) 2001, 2003, 2004 Free Software Foundation, Inc.
+ * Copyright (C) 2005 by MontaVista Software.
+ *
+ * This file is licensed under the terms of the GNU General Public License
+ * version 2. This program as licensed "as is" without any warranty of
+ * any kind, whether express or implied.
+ */
+
+#include <linux/linkage.h>
+
+#define JB_RBX 0
+#define JB_RBP 1
+#define JB_R12 2
+#define JB_R13 3
+#define JB_R14 4
+#define JB_R15 5
+#define JB_RSP 6
+#define JB_PC 7
+
+.code64
+
+/* This must be called prior to kgdb_fault_longjmp and
+ * kgdb_fault_longjmp must not be called outside of the context of the
+ * last call to kgdb_fault_setjmp.
+ */
+ENTRY(kgdb_fault_setjmp)
+/* Save registers. */
+movq %rbx, (JB_RBX*8)(%rdi)
+movq %rbp, (JB_RBP*8)(%rdi)
+movq %r12, (JB_R12*8)(%rdi)
+movq %r13, (JB_R13*8)(%rdi)
+movq %r14, (JB_R14*8)(%rdi)
+movq %r15, (JB_R15*8)(%rdi)
+leaq 8(%rsp), %rdx /* Save SP as it will be after we return. */
+movq %rdx, (JB_RSP*8)(%rdi)
+movq (%rsp), %rax /* Save PC we are returning to now. */
+movq %rax, (JB_PC*8)(%rdi)
+/* Set return value for setjmp. */

```

```

+ mov $0,%eax
+ movq (JB_PC*8)(%rdi),%rdx
+ movq (JB_RSP*8)(%rdi),%rsp
+ jmpq %%rdx
+
+ENTRY(kgdb_fault_longjmp)
+ /* Restore registers. */
+ movq (JB_RBX*8)(%rdi),%rbx
+ movq (JB_RBP*8)(%rdi),%rbp
+ movq (JB_R12*8)(%rdi),%r12
+ movq (JB_R13*8)(%rdi),%r13
+ movq (JB_R14*8)(%rdi),%r14
+ movq (JB_R15*8)(%rdi),%r15
+ /* Set return value for setjmp. */
+ movq (JB_PC*8)(%rdi),%rdx
+ movq (JB_RSP*8)(%rdi),%rsp
+ mov $1,%eax
+ jmpq %%rdx

```

Index: linux-2.6.14/arch/x86_64/kernel/Makefile

```

----- linux-2.6.14.orig/arch/x86_64/kernel/Makefile
+++ linux-2.6.14/arch/x86_64/kernel/Makefile
@@ -29,6 +29,7 @@ obj-$(CONFIG_GART_IOMMU) += pci-gart.o a
obj-$(CONFIG_DUMMY_IOMMU) += pci-nommu.o pci-dma.o
obj-$(CONFIG_SWIOTLB) += swiotlb.o
obj-$(CONFIG_KPROBES) += kprobes.o
+obj-$(CONFIG_KGDB) += kgdb.o kgdb-jmp.o
obj-$(CONFIG_X86_PM_TIMER) += pmtimer.o

```

obj-\$(CONFIG_MODULES) += module.o

Index: linux-2.6.14/include/asm-x86_64/kgdb.h

```

----- /dev/null
+++ linux-2.6.14/include/asm-x86_64/kgdb.h
@@ -0,0 +1,50 @@
+#ifdef __KERNEL__
+#ifndef _ASM_KGDB_H_
+#define _ASM_KGDB_H_
+
+
+ /*
+ * Copyright (C) 2001-2004 Amit S. Kale
+ */
+
+ /*
+ * Note that this register image is in a different order than
+ * the register image that Linux produces at interrupt time.
+ *
+ * Linux's register image is defined by struct pt_regs in ptrace.h.
+ * Just why GDB uses a different order is a historical mystery.
+ */
+#define _RAX 0

```

```

+#define _RDX 1
+#define _RCX 2
+#define _RBX 3
+#define _RSI 4
+#define _RDI 5
+#define _RBP 6
+#define _RSP 7
+#define _R8 8
+#define _R9 9
+#define _R10 10
+#define _R11 11
+#define _R12 12
+#define _R13 13
+#define _R14 14
+#define _R15 15
+#define _PC 16
+#define _PS 17
+
+/* Number of bytes of registers. */
+#define NUMREGBYTES ((_PS+1)*8)
+#define NUMCRITREGBYTES (8 * 8) /* 8 registers. */
+
+#ifndef __ASSEMBLY__
+/* BUFMAX defines the maximum number of characters in inbound/outbound
+ * buffers at least NUMREGBYTES*2 are needed for register packets, and
+ * a longer buffer is needed to list all threads. */
+#define BUFMAX 1024
+#define BREAKPOINT() asm(" int $3");
+#define CHECK_EXCEPTION_STACK() ((amp_get_cpu_var(init_tss))[0].ist[0])
+#define BREAK_INSTR_SIZE 1
+#define CACHE_FLUSH_IS_SAFE 1
+#endif /* !__ASSEMBLY__ */
+#endif /* _ASM_KGDB_H */
+#endif /* __KERNEL__ */

```

Index: linux-2.6.14/include/asm-x86_64/system.h

```

----- linux-2.6.14.orig/include/asm-x86_64/system.h
+++ linux-2.6.14/include/asm-x86_64/system.h
@@ -27,7 +27,9 @@
     , "rcx", "rbx", "rdx", "r8", "r9", "r10", "r11", "r12", "r13", "r14", "r15"

#define switch_to(prev,next,last) \
- asm volatile(SAVE_CONTEXT \
+ asm volatile(".globl __switch_to_begin\n\t" \
+ "__switch_to_begin:\n\t" \
+ SAVE_CONTEXT \
     "movq %%rsp,%P[threadrsp](%[prev])\n\t" /* save RSP */ \
     "movq %P[threadrsp](%[next]),%%rsp\n\t" /* restore RSP */ \
     "call __switch_to\n\t" \
@@ -39,6 +41,8 @@
     "movq %%rax,%%rdi\n\t" \

```

```

"jc ret_from_fork\n\t" \
RESTORE_CONTEXT \
+ ".globl __switch_to_end\n\t" \
+ "__switch_to_end:\n\t" \
: "=a" (last) \
: [next] "S" (next), [prev] "D" (prev), \
[threadrsp] "i" (offsetof(struct task_struct, thread.rsp)), \

```

Index: linux-2.6.14/lib/Kconfig.debug

```

----- linux-2.6.14.orig/lib/Kconfig.debug
+++ linux-2.6.14/lib/Kconfig.debug
@@ -187,7 +187,7 @@ config WANT_EXTRA_DEBUG_INFORMATION
config KGDB
    bool "KGDB: kernel debugging with remote gdb"
    select WANT_EXTRA_DEBUG_INFORMATION
- depends on DEBUG_KERNEL && (X86 || MIPS || IA64 || (!SMP || BROKEN) && PPC32)
+ depends on DEBUG_KERNEL && (X86 || MIPS || IA64 || X86_64 || (!SMP || BROKEN) && PPC32)
    help
    If you say Y here, it will be possible to remotely debug the
    kernel using gdb. It is strongly suggested that you enable

```

Index: linux-2.6.14/include/linux/kgdb.h

```

----- linux-2.6.14.orig/include/linux/kgdb.h
+++ linux-2.6.14/include/linux/kgdb.h
@@ -24,6 +24,10 @@
#include <linux/linkage.h>
#include <linux/init.h>

```

```

+#ifndef CHECK_EXCEPTION_STACK
+#define CHECK_EXCEPTION_STACK() 1
+#endif
+
struct tasklet_struct;
struct pt_regs;
struct task_struct;

```

Index: linux-2.6.14/kernel/kgdb.c

```

----- linux-2.6.14.orig/kernel/kgdb.c
+++ linux-2.6.14/kernel/kgdb.c
@@ -1629,7 +1629,8 @@ void kgdb_unregister_io_module(struct kg
*/
static void kgdb_tasklet_bpt(unsigned long ing)
{
- breakpoint();
+ if(CHECK_EXCEPTION_STACK())
+ breakpoint();
}

DECLARE_TASKLET(kgdb_tasklet_breakpoint, kgdb_tasklet_bpt, 0);
@@ -1640,6 +1641,17 @@ DECLARE_TASKLET(kgdb_tasklet_breakpoint,
*/

```

```
static void __init kgdb_early_entry(void)
{
+ /*
+ * Don't try and do anything until the architecture is able to
+ * setup the exception stack. In this case, it is up to the
+ * architecture to hook in and look at us when they are ready.
+ */
+ if(!CHECK_EXCEPTION_STACK()) {
+ kgdb_initialized = -1;
+ tasklet_schedule(&kgdb_tasklet_breakpoint);
+ return;
+ }
+
+ /* Let the architecture do any setup that it needs to. */
+ kgdb_arch_init();
```

--

Tom

-

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/>