```
**********************************************************
   12924 Wed Aug 15 14:55:58 2018
new/usr/src/cmd/mdb/common/kmdb/kaif_start.c
9736 kmdb tortures via single-step miscellaneous trap
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2018 Joyent, Inc.
 */

#pragma ident   "%Z%%M% %I%     %E% SMI"

/*
 * The main CPU-control loops, used to control masters and slaves.
 */

#include <sys/types.h>

#include <kmdb/kaif.h>
#include <kmdb/kaif_start.h>
#include <kmdb/kmdb_asmutil.h>
#include <kmdb/kmdb_dpi_impl.h>
#include <kmdb/kmdb_kdi.h>

#define KAIF_SLAVE_CMD_SPIN     0
#define KAIF_SLAVE_CMD_SWITCH   1
#define KAIF_SLAVE_CMD_RESUME   2
#define KAIF_SLAVE_CMD_FLUSH    3
#define KAIF_SLAVE_CMD_REBOOT   4
#if defined(__sparc)
#define KAIF_SLAVE_CMD_ACK      5
#endif


/*
 * Used to synchronize attempts to set kaif_master_cpuid.  kaif_master_cpuid may
 * be read without kaif_master_lock, and may be written by the current master
 * CPU.
 */
int kaif_master_cpuid = KAIF_MASTER_CPUID_UNSET;
static uintptr_t kaif_master_lock = 0;

/*
```



```
 * Used to ensure that all CPUs leave the debugger together. kaif_loop_lock must
 * be held to write kaif_looping, but need not be held to read it.
 */
static volatile uint_t kaif_looping;
static uintptr_t kaif_loop_lock;

static volatile int kaif_slave_cmd;
static volatile int kaif_slave_tgt;     /* target cpuid for CMD_SWITCH */

static void
kaif_lock_enter(uintptr_t *lock)
{
        while (cas(lock, 0, 1) != 0)
                continue;
        membar_producer();
}
_____unchanged_portion_omitted_

int
kaif_main_loop(kaif_cpusave_t *cpusave)
{
        int cmd;

        if (kaif_master_cpuid == KAIF_MASTER_CPUID_UNSET) {
                /*
                 * Special case: Unload requested before first debugger entry.
                 * Don't stop the world, as there's nothing to clean up that
                 * can't be handled by the running kernel.
                 */
                if (!kmdb_dpi_resume_requested &&
                    kmdb_kdi_get_unload_request()) {
                        cpusave->krs_cpu_state = KAIF_CPU_STATE_NONE;
                        return (KAIF_CPU_CMD_RESUME);
                }

                /*
                 * We're a slave with no master, so just resume.  This can
                 * happen if, prior to this, two CPUs both raced through
                 * kdi_cmnint() - for example, a breakpoint on a frequently
                 * called function.  The loser will be redirected to the slave
                 * loop; note that the event itself is lost at this point.
                 *
                 * The winner will then cross-call that slave, but it won't
                 * actually be received until the slave returns to the kernel
                 * and enables interrupts.  We'll then come back in via
                 * kdi_slave_entry() and hit this path.
                        * Special case: Unload requested before first debugger
                        * entry.  Don't stop the world, as there's nothing to
                        * clean up that can't be handled by the running kernel.
                 */
                if (cpusave->krs_cpu_state == KAIF_CPU_STATE_SLAVE) {
                        cpusave->krs_cpu_state = KAIF_CPU_STATE_NONE;
                        return (KAIF_CPU_CMD_RESUME);
                }

                kaif_select_master(cpusave);

#ifdef __sparc
                if (kaif_master_cpuid == cpusave->krs_cpu_id) {
                        /*
                         * Everyone has arrived, so we can disarm the post-PROM
                         * entry point.
                         */
                        *kaif_promexitarmp = 0;
                        membar_producer();
```

```
333                 }
334 #endif
335         } else if (kaif_master_cpuid == cpusave->krs_cpu_id) {
336                 cpusave->krs_cpu_state = KAIF_CPU_STATE_MASTER;
337         } else {
338                 cpusave->krs_cpu_state = KAIF_CPU_STATE_SLAVE;
339         }

341         cpusave->krs_cpu_flushed = 0;

343         kaif_lock_enter(&kaif_loop_lock);
344         kaif_looping++;
345         kaif_lock_exit(&kaif_loop_lock);

347         /*
348          * We know who the master and slaves are, so now they can go off
349          * to their respective loops.
350          */
351         do {
352                 if (kaif_master_cpuid == cpusave->krs_cpu_id)
353                         cmd = kaif_master_loop(cpusave);
354                 else
355                         cmd = kaif_slave_loop(cpusave);
356         } while (cmd == KAIF_CPU_CMD_SWITCH);

358         kaif_lock_enter(&kaif_loop_lock);
359         kaif_looping--;
360         kaif_lock_exit(&kaif_loop_lock);

362         cpusave->krs_cpu_state = KAIF_CPU_STATE_NONE;

364         if (cmd == KAIF_CPU_CMD_RESUME) {
365                 /*
366                  * By this point, the master has directed the slaves to resume,
367                  * and everyone is making their way to this point.  We're going
368                  * to block here until all CPUs leave the master and slave
369                  * loops.  When all have arrived, we'll turn them all loose.
370                  * This barrier is required for two reasons:
371                  *
372                  * 1. There exists a race condition whereby a CPU could reenter
373                  *    the debugger while another CPU is still in the slave loop
374                  *    from this debugger entry.  This usually happens when the
375                  *    current master releases the slaves, and makes it back to
376                  *    the world before the slaves notice the release.  The
377                  *    former master then triggers a debugger entry, and attempts
378                  *    to stop the slaves for this entry before they've even
379                  *    resumed from the last one.  When the slaves arrive here,
380                  *    they'll have re-disabled interrupts, and will thus ignore
381                  *    cross-calls until they finish resuming.
382                  *
383                  * 2. At the time of this writing, there exists a SPARC bug that
384                  *    causes an apparently unsolicited interrupt vector trap
385                  *    from OBP to one of the slaves.  This wouldn't normally be
386                  *    a problem but for the fact that the cross-called CPU
387                  *    encounters some sort of failure while in OBP.  OBP
388                  *    recovers by executing the debugger-hook word, which sends
389                  *    the slave back into the debugger, triggering a debugger
390                  *    fault.  This problem seems to only happen during resume,
391                  *    the result being that all CPUs save for the cross-called
392                  *    one make it back into the world, while the cross-called
393                  *    one is stuck at the debugger fault prompt.  Leave the
394                  *    world in that state too long, and you'll get a mondo
395                  *    timeout panic.  If we hold everyone here, we can give the
396                  *    the user a chance to trigger a panic for further analysis.
397                  *    To trigger the bug, "pool_unlock:b :c" and "while : ; do
398                  *    psrset -p ; done".
```



```
399                  *
400                  * When the second item is fixed, the barrier can move into
401                  * kaif_select_master(), immediately prior to the setting of
402                  * kaif_master_cpuid.
403                  */
404                 while (kaif_looping != 0)
405                         continue;
406         }

408         return (cmd);
409 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    2853 Wed Aug 15 14:55:58 2018
new/usr/src/uts/intel/amd64/sys/kdi_regs.h
9736 kmdb tortures via single-step miscellaneous trap
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

#ifndef _AMD64_SYS_KDI_REGS_H
#define _AMD64_SYS_KDI_REGS_H

#include <sys/stddef.h>

#ifdef __cplusplus
extern "C" {
#endif

/*
 * A modified version of struct regs layout.
 */

#define KDIREG_SAVFP    0
#define KDIREG_SAVPC    1
#define KDIREG_RDI      2
#define KDIREG_RSI      3
#define KDIREG_RDX      4
#define KDIREG_RCX      5
#define KDIREG_R8       6
#define KDIREG_R9       7
#define KDIREG_RAX      8
#define KDIREG_RBX      9
#define KDIREG_RBP      10
#define KDIREG_R10      11
#define KDIREG_R11      12
#define KDIREG_R12      13
#define KDIREG_R13      14
#define KDIREG_R14      15
#define KDIREG_R15      16
#define KDIREG_FSBASE   17
```



```
#define KDIREG_GSBASE   18
#define KDIREG_KGSBASE  19
#define KDIREG_CR2      20
#define KDIREG_CR3      21
#define KDIREG_DS       22
#define KDIREG_ES       23
#define KDIREG_FS       24
#define KDIREG_GS       25
#define KDIREG_TRAPNO   26
#define KDIREG_ERR      27
#define KDIREG_RIP      28
#define KDIREG_CS       29
#define KDIREG_RFLAGS   30
#define KDIREG_RSP      31
#define KDIREG_SS       32

#define KDIREG_NGREG    (KDIREG_SS + 1)

#define KDIREG_PC       KDIREG_RIP
#define KDIREG_SP       KDIREG_RSP
#define KDIREG_FP       KDIREG_RBP

#if !defined(_ASM)

/*
 * Handy for debugging krs_gregs; keep in sync with the KDIREG_* above.
 */
typedef struct {
        greg_t kr_savfp;
        greg_t kr_savpc;
        greg_t kr_rdi;
        greg_t kr_rsi;
        greg_t kr_rdx;
        greg_t kr_rcx;
        greg_t kr_r8;
        greg_t kr_r9;
        greg_t kr_rax;
        greg_t kr_rbx;
        greg_t kr_rbp;
        greg_t r_r10;
        greg_t r_r11;
        greg_t r_r12;
        greg_t r_r13;
        greg_t r_r14;
        greg_t r_r15;
        greg_t kr_fsbase;
        greg_t kr_gsbase;
        greg_t kr_kgsbase;
        greg_t kr_cr2;
        greg_t kr_cr3;
        greg_t kr_ds;
        greg_t kr_es;
        greg_t kr_fs;
        greg_t kr_gs;
        greg_t kr_trapno;
        greg_t kr_err;
        greg_t kr_rip;
        greg_t kr_cs;
        greg_t kr_rflags;
        greg_t kr_rsp;
        greg_t kr_ss;
} kdiregs_t;

#if defined(_KERNEL)
CTASSERT(offsetof(kdiregs_t, kr_ss) == ((KDIREG_NGREG - 1) * sizeof (greg_t)));
#endif
```

```
127 #endif /* !_ASM */

129 #ifdef __cplusplus
130 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   18997 Wed Aug 15 14:55:59 2018
new/usr/src/uts/intel/kdi/kdi_asm.s
9736 kmdb tortures via single-step miscellaneous trap
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
______unchanged_portion_omitted_

 370 /*
 371  * The cross-call handler for slave CPUs.
 372  *
 373  * The debugger is single-threaded, so only one CPU, called the master, may be
 374  * running it at any given time.  The other CPUs, known as slaves, spin in a
 375  * busy loop until there's something for them to do.  This is the entry point
 376  * for the slaves - they'll be sent here in response to a cross-call sent by the
 377  * master.
 378  */

 380         ENTRY_NP(kdi_slave_entry)

 382         /*
 383          * Cross calls are implemented as function calls, so our stack currently
 384          * looks like one you'd get from a zero-argument function call.  That
 385          * is, there's the return %rip at %rsp, and that's about it.  We need
 386          * to make it look like an interrupt stack.  When we first save, we'll
 387          * reverse the saved %ss and %rip, which we'll fix back up when we've
 388          * freed up some general-purpose registers.  We'll also need to fix up
 389          * the saved %rsp.
 390          */

 392         pushq   %rsp            /* pushed value off by 8 */
 393         pushfq
 394         CLI(%rax)
 395         pushq   $KCS_SEL
 396         clrq    %rax
 397         movw    %ss, %ax
 398         pushq   %rax            /* rip should be here */
 399         pushq   $-1             /* phony trap error code */
 400         pushq   $-1             /* phony trap number */

 402         subq    $REG_OFF(KDIREG_TRAPNO), %rsp
 403         KDI_SAVE_REGS(%rsp)

 405         movq    %cr3, %rax
 406         movq    %rax, REG_OFF(KDIREG_CR3)(%rsp)

 408         movq    REG_OFF(KDIREG_SS)(%rsp), %rax
 409         movq    %rax, REG_OFF(KDIREG_SAVPC)(%rsp)
 410         xchgq   REG_OFF(KDIREG_RIP)(%rsp), %rax
 411         movq    %rax, REG_OFF(KDIREG_SS)(%rsp)

 413         movq    REG_OFF(KDIREG_RSP)(%rsp), %rax
 414         addq    $8, %rax
 415         movq    %rax, REG_OFF(KDIREG_RSP)(%rsp)

 417         /*
 418          * We've saved all of the general-purpose registers, and have a stack
 419          * that is irettable (after we strip down to the error code)
 420          */

 422         GET_CPUSAVE_ADDR        /* %rax = cpusave, %rbx = CPU ID */

 424         ADVANCE_CRUMB_POINTER(%rax, %rcx, %rdx)

 426         ADD_CRUMB(%rax, KRM_CPU_STATE, $KDI_CPU_STATE_SLAVE, %rdx)
```



```
 428         movq    REG_OFF(KDIREG_RIP)(%rsp), %rcx
 429         ADD_CRUMB(%rax, KRM_PC, %rcx, %rdx)
 430         movq    REG_OFF(KDIREG_RSP)(%rsp), %rcx
 431         ADD_CRUMB(%rax, KRM_SP, %rcx, %rdx)
 432         ADD_CRUMB(%rax, KRM_TRAPNO, $-1, %rdx)

 434         movq    $KDI_CPU_STATE_SLAVE, KRS_CPU_STATE(%rax)

 436         pushq   %rax
 437         jmp     kdi_save_common_state

 439         SET_SIZE(kdi_slave_entry)
______unchanged_portion_omitted_
```

```
**********************************************************
   12331 Wed Aug 15 14:55:59 2018
new/usr/src/uts/intel/kdi/kdi_idt.c
9736 kmdb tortures via single-step miscellaneous trap
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

/*
 * Management of KMDB's IDT, which is installed upon KMDB activation.
 *
 * Debugger activation has two flavors, which cover the cases where KMDB is
 * loaded at boot, and when it is loaded after boot.  In brief, in both cases,
 * the KDI needs to interpose upon several handlers in the IDT.  When
 * mod-loaded KMDB is deactivated, we undo the IDT interposition, restoring the
 * handlers to what they were before we started.
 *
 * We also take over the entirety of IDT (except the double-fault handler) on
 * the active CPU when we're in kmdb so we can handle things like page faults
 * sensibly.
 *
 * Boot-loaded KMDB
 *
 * When we're first activated, we're running on boot's IDT.  We need to be able
 * to function in this world, so we'll install our handlers into boot's IDT.
 * This is a little complicated: we're using the fake cpu_t set up by
 * boot_kdi_tmpinit(), so we can't access cpu_idt directly.  Instead,
 * kdi_idt_write() notices that cpu_idt is NULL, and works around this problem.
 *
 * Later, when we're about to switch to the kernel's IDT, it'll call us via
 * kdi_idt_sync(), allowing us to add our handlers to the new IDT.  While
 * boot-loaded KMDB can't be unloaded, we still need to save the descriptors we
 * replace so we can pass traps back to the kernel as necessary.
 *
 * The last phase of boot-loaded KMDB activation occurs at non-boot CPU
 * startup.  We will be called on each non-boot CPU, thus allowing us to set up
 * any watchpoints that may have been configured on the boot CPU and interpose
 * on the given CPU's IDT.  We don't save the interposed descriptors in this
 * case -- see kdi_cpu_init() for details.
 *
```



```
 * Mod-loaded KMDB
 *
 * This style of activation is much simpler, as the CPUs are already running,
 * and are using their own copy of the kernel's IDT.  We simply interpose upon
 * each CPU's IDT.  We save the handlers we replace, both for deactivation and
 * for passing traps back to the kernel.  Note that for the hypervisors'
 * benefit, we need to xcall to the other CPUs to do this, since we need to
 * actively set the trap entries in its virtual IDT from that vcpu's context
 * rather than just modifying the IDT table from the CPU running kdi_activate().
 */

#include <sys/types.h>
#include <sys/segments.h>
#include <sys/trap.h>
#include <sys/cpuvar.h>
#include <sys/reboot.h>
#include <sys/sunddi.h>
#include <sys/archsystm.h>
#include <sys/kdi_impl.h>
#include <sys/x_call.h>
#include <ia32/sys/psw.h>
#include <vm/hat_i86.h>

#define KDI_GATE_NVECS  3

#define KDI_IDT_NOSAVE  0
#define KDI_IDT_SAVE    1

#define KDI_IDT_DTYPE_KERNEL    0
#define KDI_IDT_DTYPE_BOOT      1

/* Solely to keep kdiregs_t in the CTF, otherwise unused. */
kdiregs_t kdi_regs;

kdi_cpusave_t *kdi_cpusave;
int kdi_ncpusave;

static kdi_main_t kdi_kmdb_main;

kdi_drreg_t kdi_drreg;

#ifndef __amd64
/* Used to track the current set of valid kernel selectors. */
uint32_t        kdi_cs;
uint32_t        kdi_ds;
uint32_t        kdi_fs;
uint32_t        kdi_gs;
#endif

uintptr_t       kdi_kernel_handler;

int             kdi_trap_switch;

#define KDI_MEMRANGES_MAX       2

kdi_memrange_t  kdi_memranges[KDI_MEMRANGES_MAX];
int             kdi_nmemranges;

typedef void idt_hdlr_f(void);

extern idt_hdlr_f kdi_trap0, kdi_trap1, kdi_int2, kdi_trap3, kdi_trap4;
extern idt_hdlr_f kdi_trap5, kdi_trap6, kdi_trap7, kdi_trap9;
extern idt_hdlr_f kdi_traperr10, kdi_traperr11, kdi_traperr12;
extern idt_hdlr_f kdi_traperr13, kdi_traperr14, kdi_trap16, kdi_traperr17;
extern idt_hdlr_f kdi_trap18, kdi_trap19, kdi_trap20, kdi_ivct32;
extern idt_hdlr_f kdi_invaltrap;
```

```
 126 extern size_t kdi_ivct_size;

 128 typedef struct kdi_gate_spec {
 129         uint_t kgs_vec;
 130         uint_t kgs_dpl;
 131 } kdi_gate_spec_t;
```

_____***unchanged_portion_omitted_***