```
**********************************************************
    2524 Fri Apr  6 17:24:52 2018
new/usr/src/cmd/mdb/Makefile.kmdb.files
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
#
# 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 (c) 2012 by Delphix. All rights reserved.
# Copyright (c) 2018 Joyent, Inc. All rights reserved.
# Copyright (c) 2012 Joyent, Inc. All rights reserved.
#

KMDBSRCS += \
        ffs.c \
        kaif_start.c \
        mdb.c \
        mdb_addrvec.c \
        mdb_argvec.c \
        mdb_callb.c \
        mdb_cmdbuf.c \
        mdb_cmds.c \
        kvm_cpu.c \
        kmdb_conf.c \
        kmdb_context.c \
        kmdb_create.c \
        mdb_ctf.c \
        kmdb_ctf_open.c \
        mdb_debug.c \
        kmdb_demangle.c \
        mdb_disasm.c \
        kmdb_dl.c \
        kmdb_dpi.c \
        mdb_dump.c \
        mdb_err.c \
        mdb_evset.c \
        kmdb_fault.c \
        kmdb_fdio.c \
        mdb_fmt.c \
```



```
        mdb_frame.c \
        mdb_gelf.c \
        mdb_help.c \
        mdb_io.c \
        kmdb_kdi.c \
        kmdb_kvm.c \
        mdb_logio.c \
        mdb_list.c \
        mdb_macalias.c \
        kmdb_main.c \
        mdb_modapi.c \
        mdb_module.c \
        kmdb_module.c \
        kmdb_module_load.c \
        mdb_nm.c \
        mdb_nv.c \
        mdb_pipeio.c \
        mdb_print.c \
        kmdb_promio.c \
        kmdb_promif.c \
        mdb_set.c \
        kmdb_shell.c \
        mdb_signal.c \
        mdb_string.c \
        mdb_strio.c \
        kmdb_stubs.c \
        mdb_tab.c \
        mdb_target.c \
        kmdb_terminfo.c \
        mdb_termio.c \
        mdb_typedef.c \
        mdb_umem.c \
        kmdb_umemglue.c \
        mdb_value.c \
        mdb_vcb.c \
        mdb_wcb.c \
        mdb_whatis.c \
        kmdb_wr.c

KMDBML +=

KMDBOBJS = $(KMDBSRCS:%.c=%.o) $(KMDBML:%.s=%.o)

PROMSRCS +=

PROMOBJS = $(PROMSRCS:%.c=%.o)

KCTLSRCS += \
        kctl_auxv.c \
        kctl_dmod.c \
        kctl_err.c \
        kctl_main.c \
        kctl_mod.c \
        kctl_string.c \
        kctl_wr.c

KCTLML +=

KCTLOBJS = $(KCTLSRCS:%.c=%.o) $(KCTLML:%.s=%.o)

SRCS += $(KMDBSRCS) $(PROMSRCS)
MLSRCS += $(KMDBML)
OBJS = $(SRCS:%.c=%.o) $(KMDBML:%.s=%.o)

ALLOBJS = $(OBJS) $(KCTLOBJS)
ALLLINTFILES = $(ALLOBJS:%.o=%.ln)
```

```
123 # files that need KMDB_VERSION defined
124 VERSFILES = \
125         kmdb_conf.c \
126         kctl_main.c
127 VERSOBJS = $(VERSFILES:%.c=%.o)
```

```
**********************************************************
    3496 Fri Apr  6 17:24:52 2018
new/usr/src/cmd/mdb/common/kmdb/kmdb_dpi_impl.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  *
  25  * Copyright 2018 Joyent, Inc.
  26  */

  28 #ifndef _KMDB_DPI_IMPL_H
  29 #define _KMDB_DPI_IMPL_H

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

  31 #include <setjmp.h>
  32 #ifdef  __sparc
  33 #include <sys/regset.h>
  34 #endif  /* __sparc */
  35 #include <sys/types.h>

  37 #include <kmdb/kmdb_auxv.h>
  38 #include <kmdb/kmdb_dpi.h>
  39 #include <mdb/mdb_kreg.h>
  40 #include <mdb/mdb_target.h>

  42 #ifdef __cplusplus
  43 extern "C" {
  44 #endif

  46 extern jmp_buf *kmdb_dpi_fault_pcb;

  48 /*
  49  * The routines used by the kmdb side of the DPI to access the saved state
  50  * of the current kernel instance, and to control that instance.  A populated
  51  * version of this vector is provided by the DPI backend used to control the
  52  * machine.  General use of the kmdb DPI is not via direct invocation of the
  53  * functions in this ops vector, but rather flows through the convenience
  54  * wrappers in kmdb_dpi.c.
  55  */
```



```
  56 struct dpi_ops {
  57         int (*dpo_init)(kmdb_auxv_t *);

  59         void (*dpo_debugger_activate)(kdi_debugvec_t **, uint_t);
  60         void (*dpo_debugger_deactivate)(void);

  62         void (*dpo_enter_mon)(void);

  64         void (*dpo_modchg_register)(void (*)(struct modctl *, int));
  65         void (*dpo_modchg_cancel)(void);

  67         int (*dpo_get_cpu_state)(int);
  68         int (*dpo_get_master_cpuid)(void);

  70         const mdb_tgt_gregset_t *(*dpo_get_gregs)(int);
  71         int (*dpo_get_register)(const char *, kreg_t *);
  72         int (*dpo_set_register)(const char *, kreg_t);
  73 #ifdef __sparc
  74         int (*dpo_get_rwin)(int, int, struct rwindow *);
  75         int (*dpo_get_nwin)(int);
  76 #endif

  78         int (*dpo_brkpt_arm)(uintptr_t, mdb_instr_t *);
  79         int (*dpo_brkpt_disarm)(uintptr_t, mdb_instr_t);

  81         int (*dpo_wapt_validate)(kmdb_wapt_t *);
  82         int (*dpo_wapt_reserve)(kmdb_wapt_t *);
  83         void (*dpo_wapt_release)(kmdb_wapt_t *);
  84         void (*dpo_wapt_arm)(kmdb_wapt_t *);
  85         void (*dpo_wapt_disarm)(kmdb_wapt_t *);
  86         int (*dpo_wapt_match)(kmdb_wapt_t *);

  88         int (*dpo_step)(void);
  89 #if defined(__i386) || defined(__amd64)
  90         void (*dpo_step_branch)(void);
  91 #endif

  90         uintptr_t (*dpo_call)(uintptr_t, uint_t, const uintptr_t *);

  92         void (*dpo_dump_crumbs)(uintptr_t, int);

  97 #if defined(__i386) || defined(__amd64)
  98         void (*dpo_msr_add)(const kdi_msr_t *);
  99         uint64_t (*dpo_msr_get)(int, uint_t);
 100 #endif

  94 #ifdef __sparc
  95         void (*dpo_kernpanic)(int);
  96 #endif
  97 };
______unchanged_portion_omitted_
```

```
**********************************************************
   63016 Fri Apr  6 17:24:53 2018
new/usr/src/cmd/mdb/common/kmdb/kmdb_kvm.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 *
 * Copyright 2018 Joyent, Inc.
 */

#include <kmdb/kmdb_kvm.h>
#include <kmdb/kvm.h>
#include <kmdb/kmdb_kdi.h>
#include <kmdb/kmdb_promif.h>
#include <kmdb/kmdb_module.h>
#include <kmdb/kmdb_asmutil.h>
#include <mdb/mdb_types.h>
#include <mdb/mdb_conf.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_string.h>
#include <mdb/mdb_ctf.h>
#include <mdb/mdb_kreg_impl.h>
#include <mdb/mdb_ks.h>
#include <mdb/mdb.h>

#include <strings.h>
#include <dlfcn.h>
#include <sys/isa_defs.h>
#include <sys/kobj.h>
#include <sys/kobj_impl.h>
#include <sys/bitmap.h>
#include <vm/as.h>

static const char KMT_RTLD_NAME[] = "krtld";
static const char KMT_MODULE[] = "mdb_ks";
static const char KMT_CTFPARENT[] = "genunix";
```



```
static mdb_list_t kmt_defbp_list;       /* List of current deferred bp's */
static int kmt_defbp_lock;              /* For list, running kernel holds */
static uint_t kmt_defbp_modchg_isload;  /* Whether mod change is load/unload */
static struct modctl *kmt_defbp_modchg_modctl; /* modctl for defbp checking */
static uint_t kmt_defbp_num;            /* Number of referenced def'd bp's */
static int kmt_defbp_bpspec;            /* vespec for def'd bp activation bp */

static const mdb_se_ops_t kmt_brkpt_ops;
static const mdb_se_ops_t kmt_wapt_ops;

static void kmt_sync(mdb_tgt_t *);

typedef struct kmt_symarg {
        mdb_tgt_sym_f *sym_cb;          /* Caller's callback function */
        void *sym_data;                 /* Callback function argument */
        uint_t sym_type;                /* Symbol type/binding filter */
        mdb_syminfo_t sym_info;         /* Symbol id and table id */
        const char *sym_obj;            /* Containing object */
} kmt_symarg_t;
_____unchanged_portion_omitted_

/*ARGSUSED*/
static int
kmt_status_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
{
        kmt_data_t *kmt = mdb.m_target->t_data;
        struct utsname uts;
        char uuid[37];
        kreg_t tt;

        if (mdb_tgt_readsym(mdb.m_target, MDB_TGT_AS_VIRT, &uts, sizeof (uts),
            "unix", "utsname") != sizeof (uts)) {
                warn("failed to read 'utsname' struct from kernel\n");
                bzero(&uts, sizeof (uts));
                (void) strcpy(uts.nodename, "unknown machine");
        }

        mdb_printf("debugging live kernel (%d-bit) on %s\n",
            (int)(sizeof (void *) * NBBY),
            (*uts.nodename == '\0' ? "(not set)" : uts.nodename));
        mdb_printf("operating system: %s %s (%s)\n",
            uts.release, uts.version, uts.machine);

        if (mdb_tgt_readsym(mdb.m_target, MDB_TGT_AS_VIRT, uuid, sizeof (uuid),
            "genunix", "dump_osimage_uuid") != sizeof (uuid)) {
                warn("failed to read 'dump_osimage_uuid' string from kernel\n");
                (void) strcpy(uuid, "(error)");
        } else if (*uuid == '\0') {
                (void) strcpy(uuid, "(not set)");
        } else if (uuid[36] != '\0') {
                (void) strcpy(uuid, "(invalid)");
        }
        mdb_printf("image uuid: %s\n", uuid);

        if (kmt->kmt_cpu != NULL) {
                mdb_printf("CPU-specific support: %s\n",
                    kmt_cpu_name(kmt->kmt_cpu));
        }

        mdb_printf("DTrace state: %s\n", (kmdb_kdi_dtrace_get_state() ==
            KDI_DTSTATE_DTRACE_ACTIVE ? "active (debugger breakpoints cannot "
            "be armed)" : "inactive"));

        (void) kmdb_dpi_get_register("tt", &tt);
        mdb_printf("stopped on: %s\n", kmt_trapname(tt));
```

```
 589         (void) kmt_dmod_status("pending dmod loads:", KMDB_MC_STATE_LOADING);
 590         (void) kmt_dmod_status("pending dmod unloads:",
 591             KMDB_MC_STATE_UNLOADING);

 593         return (DCMD_OK);
 594 }
______unchanged_portion_omitted_

2373 static void
2374 kmt_destroy(mdb_tgt_t *t)
2375 {
2376         kmt_data_t *kmt = t->t_data;
2377         kmt_module_t *km, *pkm;

2379         mdb_nv_destroy(&kmt->kmt_modules);
2380         for (km = mdb_list_prev(&kmt->kmt_modlist); km != NULL; km = pkm) {
2381                 pkm = mdb_list_prev(km);
2382                 mdb_free(km, sizeof (kmt_module_t));
2383         }

2385         if (!kmt_defbp_lock)
2386                 kmt_defbp_destroy_all();

2388         if (kmt->kmt_trapmap != NULL)
2389                 mdb_free(kmt->kmt_trapmap, BT_SIZEOFMAP(kmt->kmt_trapmax));

2395         if (kmt->kmt_cpu != NULL)
2396                 kmt_cpu_destroy(kmt->kmt_cpu);

2391         if (kmt != NULL)
2392                 mdb_free(kmt, sizeof (kmt_data_t));
2393 }

2395 static const mdb_tgt_ops_t kmt_ops = {
2396         kmt_setflags,                           /* t_setflags */
2397         (int (*)()) mdb_tgt_notsup,             /* t_setcontext */
2398         kmt_activate,                           /* t_activate */
2399         (void (*)()) mdb_tgt_nop,               /* t_deactivate */
2400         kmt_periodic,                           /* t_periodic */
2401         kmt_destroy,                            /* t_destroy */
2402         kmt_name,                               /* t_name */
2403         (const char *(*)()) mdb_conf_isa,       /* t_isa */
2404         kmt_platform,                           /* t_platform */
2405         kmt_uname,                              /* t_uname */
2406         kmt_dmodel,                             /* t_dmodel */
2407         (ssize_t (*)()) mdb_tgt_notsup,         /* t_aread */
2408         (ssize_t (*)()) mdb_tgt_notsup,         /* t_awrite */
2409         kmt_read,                               /* t_vread */
2410         kmt_write,                              /* t_vwrite */
2411         kmt_pread,                              /* t_pread */
2412         kmt_pwrite,                             /* t_pwrite */
2413         kmt_read,                               /* t_fread */
2414         kmt_write,                              /* t_fwrite */
2415         kmt_ioread,                             /* t_ioread */
2416         kmt_iowrite,                            /* t_iowrite */
2417         kmt_vtop,                               /* t_vtop */
2418         kmt_lookup_by_name,                     /* t_lookup_by_name */
2419         kmt_lookup_by_addr,                     /* t_lookup_by_addr */
2420         kmt_symbol_iter,                        /* t_symbol_iter */
2421         kmt_mapping_iter,                       /* t_mapping_iter */
2422         kmt_object_iter,                        /* t_object_iter */
2423         kmt_addr_to_map,                        /* t_addr_to_map */
2424         kmt_name_to_map,                        /* t_name_to_map */
2425         kmt_addr_to_ctf,                        /* t_addr_to_ctf */
2426         kmt_name_to_ctf,                        /* t_name_to_ctf */
2427         kmt_status,                             /* t_status */
```



```
2428         (int (*)()) mdb_tgt_notsup,             /* t_run */
2429         kmt_step,                               /* t_step */
2430         kmt_step_out,                           /* t_step_out */
2438         kmt_step_branch,                        /* t_step_branch */
2431         kmt_next,                               /* t_next */
2432         kmt_continue,                           /* t_cont */
2433         (int (*)()) mdb_tgt_notsup,             /* t_signal */
2434         kmt_add_vbrkpt,                         /* t_add_vbrkpt */
2435         kmt_add_sbrkpt,                         /* t_add_sbrkpt */
2436         kmt_add_pwapt,                          /* t_add_pwapt */
2437         kmt_add_vwapt,                          /* t_add_vwapt */
2438         kmt_add_iowapt,                         /* t_add_iowapt */
2439         (int (*)()) mdb_tgt_null,               /* t_add_sysenter */
2440         (int (*)()) mdb_tgt_null,               /* t_add_sysexit */
2441         (int (*)()) mdb_tgt_null,               /* t_add_signal */
2442         kmt_add_trap,                           /* t_add_fault */
2443         kmt_getareg,                            /* t_getareg */
2444         kmt_putareg,                            /* t_putareg */
2445         (int (*)()) mdb_tgt_nop,                /* XXX t_stack_iter */
2446         (int (*)()) mdb_tgt_notsup              /* t_auxv */
2447 };

2449 /*
2450  * Called immediately upon resumption of the system after a step or continue.
2451  * Allows us to synchronize kmt's view of the world with reality.
2452  */
2453 /*ARGSUSED*/
2454 static void
2455 kmt_sync(mdb_tgt_t *t)
2456 {
2457         kmt_data_t *kmt = t->t_data;
2458         int symavail;

2460         mdb_dprintf(MDB_DBG_KMOD, "synchronizing with kernel\n");

2462         symavail = kmt->kmt_symavail;
2463         kmt->kmt_symavail = FALSE;

2465         /*
2466          * Resync our view of the world if the modules have changed, or if we
2467          * didn't have any symbols coming into this function.  The latter will
2468          * only happen on startup.
2469          */
2470         if (kmdb_kdi_mods_changed() || !symavail)
2471                 kmt_modlist_update(t);

2473         /*
2474          * It would be nice if we could run this less frequently, perhaps
2475          * after a dvec-initiated trigger.
2476          */
2477         kmdb_module_sync();

2479         kmt->kmt_symavail = TRUE;

2481         mdb_dprintf(MDB_DBG_KMOD, "synchronization complete\n");

2483         kmt_defbp_prune();

2485         if (kmt_defbp_num > 0 && kmt_defbp_bpspec == 0 &&
2486             kmdb_kdi_dtrace_get_state() != KDI_DTSTATE_DTRACE_ACTIVE) {
2487                 /*
2488                  * Deferred breakpoints were created while DTrace was active,
2489                  * and consequently the deferred breakpoint enabling mechanism
2490                  * wasn't activated.  Activate it now, and then try to activate
2491                  * the deferred breakpoints.  We do this so that we can catch
2492                  * the ones which may apply to modules that have been loaded
```

```
2493                  * while they were waiting for DTrace to deactivate.
2494                  */
2495                 (void) kmt_defbp_activate(t);
2496                 (void) mdb_tgt_sespec_activate_all(t);
2497         }

2507         if (kmt->kmt_cpu_retry && ((kmt->kmt_cpu = kmt_cpu_create(t)) !=
2508             NULL || errno != EAGAIN))
2509                 kmt->kmt_cpu_retry = FALSE;

2499         (void) mdb_tgt_status(t, &t->t_status);
2500 }

2502 /*
2503  * This routine executes while the kernel is running.
2504  */
2505 /*ARGSUSED*/
2506 int
2507 kmdb_kvm_create(mdb_tgt_t *t, int argc, const char *argv[])
2508 {
2509         kmt_data_t *kmt;

2511         if (argc != 0)
2512                 return (set_errno(EINVAL));

2514         kmt = mdb_zalloc(sizeof (kmt_data_t), UM_SLEEP);
2515         t->t_data = kmt;
2516         t->t_ops = &kmt_ops;
2517         t->t_flags |= MDB_TGT_F_RDWR;   /* kmdb is always r/w */

2519         (void) mdb_nv_insert(&mdb.m_nv, "cpuid", &kmt_cpuid_disc, 0,
2520             MDB_NV_PERSIST | MDB_NV_RDONLY);

2522         (void) mdb_nv_create(&kmt->kmt_modules, UM_SLEEP);

2524         kmt_init_isadep(t);

2526         kmt->kmt_symavail = FALSE;
2539         kmt->kmt_cpu_retry = TRUE;

2528         bzero(&kmt_defbp_list, sizeof (mdb_list_t));

2530         return (0);

2532 create_err:
2533         kmt_destroy(t);

2535         return (-1);
2536 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    4411 Fri Apr  6 17:24:53 2018
new/usr/src/cmd/mdb/common/kmdb/kvm.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 _KVM_H
#define _KVM_H

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

/*
 * The kmdb target
 */

#include <mdb/mdb_modapi.h>
#include <mdb/mdb_target.h>
#include <kmdb/kmdb_dpi.h>
#include <kmdb/kvm_isadep.h>
#include <kmdb/kvm_cpu.h>

#include <sys/kobj.h>

#ifdef __cplusplus
extern "C" {
#endif

#define KM_F_PRIMARY            1

#define KMT_TRAP_NOTENUM        -1      /* Glob for unnamed traps */
#define KMT_TRAP_ALL            -2      /* Glob for all traps */

typedef struct kmt_module {
        mdb_list_t      km_list;        /* List forward/back pointers */
        char            *km_name;       /* Module name */
        char            km_seen;
```



```
        GElf_Ehdr       km_ehdr;

        mdb_gelf_symtab_t *km_symtab;
        Shdr            km_symtab_hdr;
        Shdr            km_strtab_hdr;
        const void      *km_symtab_va;
        const void      *km_strtab_va;

        uintptr_t       km_text_va;
        size_t          km_text_size;
        uintptr_t       km_data_va;
        size_t          km_data_size;
        uintptr_t       km_bss_va;
        size_t          km_bss_size;
        const void      *km_ctf_va;
        size_t          km_ctf_size;

        ctf_file_t      *km_ctfp;
        struct modctl   km_modctl;
        struct module   km_module;
        int             km_flags;
} kmt_module_t;

typedef struct kmt_data {
        const mdb_tgt_regdesc_t *kmt_rds;       /* Register description table */
        mdb_nv_t        kmt_modules;            /* Hash table of modules */
        mdb_list_t      kmt_modlist;            /* List of mods in load order */
        const char      *kmt_rtld_name;         /* Module containing krtld */
        caddr_t         kmt_writemap;           /* Used to map PAs for writes */
        size_t          kmt_writemapsz;         /* Size of same */
        mdb_map_t       kmt_map;                /* Persistant map for callers */
        ulong_t         *kmt_trapmap;
        size_t          kmt_trapmax;
        kmt_cpu_t       *kmt_cpu;               /* CPU-specific plugin */
        int             kmt_cpu_retry;          /* Try CPU detect again? */
        int             kmt_symavail;           /* Symbol resolution allowed */
        uint_t          kmt_narmedbpts;         /* Number of armed brkpts */
#if defined(__i386) || defined(__amd64)
        struct {
                GElf_Sym        _kmt_cmnint;
                GElf_Sym        _kmt_cmntrap;
                GElf_Sym        _kmt_sysenter;
                GElf_Sym        _kmt_brand_sysenter;
#if defined(__amd64)
                GElf_Sym        _kmt_syscall;
                GElf_Sym        _kmt_brand_syscall;
#endif
        } kmt_intrsyms;
#endif
} kmt_data_t;
______unchanged_portion_omitted_

extern void kmt_printregs(const mdb_tgt_gregset_t *gregs);

extern const char *kmt_def_dismode(void);

extern void kmt_init_isadep(mdb_tgt_t *);
extern void kmt_startup_isadep(mdb_tgt_t *);

extern ssize_t kmt_write(mdb_tgt_t *, const void *, size_t, uintptr_t);
extern ssize_t kmt_pwrite(mdb_tgt_t *, const void *, size_t, physaddr_t);
extern ssize_t kmt_rw(mdb_tgt_t *, void *, size_t, uint64_t,
    ssize_t (*)(void *, size_t, uint64_t));
extern ssize_t kmt_writer(void *, size_t, uint64_t);
extern ssize_t kmt_ioread(mdb_tgt_t *, void *, size_t, uintptr_t);
extern ssize_t kmt_iowrite(mdb_tgt_t *, const void *, size_t, uintptr_t);
```

```
 142 extern int kmt_step_out(mdb_tgt_t *, uintptr_t *);
 146 extern int kmt_step_branch(mdb_tgt_t *);
 143 extern int kmt_next(mdb_tgt_t *, uintptr_t *);

 145 extern int kmt_stack(uintptr_t, uint_t, int, const mdb_arg_t *);
 146 extern int kmt_stackv(uintptr_t, uint_t, int, const mdb_arg_t *);
 147 extern int kmt_stackr(uintptr_t, uint_t, int, const mdb_arg_t *);
 148 extern int kmt_cpustack(uintptr_t, uint_t, int, const mdb_arg_t *, int, int);

 150 extern const char *kmt_trapname(int);

 152 #ifdef __cplusplus
 153 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   84154 Fri Apr  6 17:24:53 2018
new/usr/src/cmd/mdb/common/mdb/mdb_cmds.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 (c) 2012 by Delphix. All rights reserved.
 * Copyright (c) 2018 Joyent, Inc. All rights reserved.
 * Copyright (c) 2015 Joyent, Inc. All rights reserved.
 * Copyright (c) 2013 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 * Copyright (c) 2015, 2017 by Delphix. All rights reserved.
 */

#include <sys/elf.h>
#include <sys/elf_SPARC.h>

#include <libproc.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <errno.h>
#include <alloca.h>
#include <libctf.h>
#include <ctype.h>

#include <mdb/mdb_string.h>
#include <mdb/mdb_argvec.h>
#include <mdb/mdb_nv.h>
#include <mdb/mdb_fmt.h>
#include <mdb/mdb_target.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_conf.h>
#include <mdb/mdb_module.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_stdlib.h>
```



```
#include <mdb/mdb_lex.h>
#include <mdb/mdb_io_impl.h>
#include <mdb/mdb_help.h>
#include <mdb/mdb_disasm.h>
#include <mdb/mdb_frame.h>
#include <mdb/mdb_evset.h>
#include <mdb/mdb_print.h>
#include <mdb/mdb_nm.h>
#include <mdb/mdb_set.h>
#include <mdb/mdb_demangle.h>
#include <mdb/mdb_ctf.h>
#include <mdb/mdb_whatis.h>
#include <mdb/mdb_whatis_impl.h>
#include <mdb/mdb_macalias.h>
#include <mdb/mdb_tab.h>
#include <mdb/mdb_typedef.h>
#ifdef _KMDB
#include <kmdb/kmdb_kdi.h>
#endif
#include <mdb/mdb.h>

#ifdef __sparc
#define SETHI_MASK      0xc1c00000
#define SETHI_VALUE     0x01000000

#define IS_SETHI(machcode)      (((machcode) & SETHI_MASK) == SETHI_VALUE)

#define OP(machcode)    ((machcode) >> 30)
#define OP3(machcode)   (((machcode) >> 19) & 0x3f)
#define RD(machcode)    (((machcode) >> 25) & 0x1f)
#define RS1(machcode)   (((machcode) >> 14) & 0x1f)
#define I(machcode)     (((machcode) >> 13) & 0x01)

#define IMM13(machcode) ((machcode) & 0x1fff)
#define IMM22(machcode) ((machcode) & 0x3fffff)

#define OP_ARITH_MEM_MASK       0x2
#define OP_ARITH                0x2
#define OP_MEM                  0x3

#define OP3_CC_MASK             0x10
#define OP3_COMPLEX_MASK        0x20

#define OP3_ADD                 0x00
#define OP3_OR                  0x02
#define OP3_XOR                 0x03

#ifndef R_O7
#define R_O7    0xf
#endif
#endif /* __sparc */

static mdb_tgt_addr_t
write_uint8(mdb_tgt_as_t as, mdb_tgt_addr_t addr, uint64_t ull, uint_t rdback)
{
        uint8_t o, n = (uint8_t)ull;

        if (rdback && mdb_tgt_aread(mdb.m_target, as, &o, sizeof (o),
            addr) == -1)
                return (addr);

        if (mdb_tgt_awrite(mdb.m_target, as, &n, sizeof (n), addr) == -1)
                return (addr);

        if (rdback) {
                if (mdb_tgt_aread(mdb.m_target, as, &n, sizeof (n), addr) == -1)
```

```
 123                         return (addr);

 125                 mdb_iob_printf(mdb.m_out, "%-#*lla%16T%-#8x=%8T0x%x\n",
 126                     mdb_iob_getmargin(mdb.m_out), addr, o, n);
 127         }

 129         return (addr + sizeof (n));
 130 }
______unchanged_portion_omitted_

2734 static int
2735 cmd_step(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
2736 {
2737         int (*func)(mdb_tgt_t *, mdb_tgt_status_t *) = &mdb_tgt_step;
2738         const char *name = "single-step";

2740         if (argc > 0 && argv->a_type == MDB_TYPE_STRING) {
2741                 if (strcmp(argv->a_un.a_str, "out") == 0) {
2742                         func = &mdb_tgt_step_out;
2743                         name = "step (out)";
2744                         argv++;
2745                         argc--;
2746                 } else if (strcmp(argv->a_un.a_str, "branch") == 0) {
2747                         func = &mdb_tgt_step_branch;
2748                         name = "step (branch)";
2749                         argv++;
2750                         argc--;
2746                 } else if (strcmp(argv->a_un.a_str, "over") == 0) {
2747                         func = &mdb_tgt_next;
2748                         name = "step (over)";
2749                         argv++;
2750                         argc--;
2751                 }
2752         }

2754         return (cmd_cont_common(addr, flags, argc, argv, func, name));
2755 }
______unchanged_portion_omitted_
```

```
**********************************************************
   30681 Fri Apr  6 17:24:54 2018
new/usr/src/cmd/mdb/common/mdb/mdb_kproc.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

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

/*
 * Kernel Process View Target
 *
 * The kproc target is activated when the user is debugging a kernel using the
 * kvm target and executes a ::context dcmd to change the debugger view to one
 * of the running processes.  The kvm target's t_setcontext operation will
 * create and activate a kproc target in response to this call.  The kproc
 * target itself is built upon the kvm target's libkvm cookie and the ability
 * to read information from the kernel itself and the ability to read the
 * address space of a particular user process with kvm_aread().  It also relies
 * on a special set of functions provided by the kvm target's mdb_ks support
 * module in order to bootstrap: specifically, given the initial proc pointer,
 * mdb_ks provides functions to return the set of address space mappings, the
 * address space pointer itself, the aux vector vector saved in the u-area,
 * and the process data model.  The kproc target maintains a list of address
 * space mappings (kp_map_t) and load objects (kp_file_t), and for each load
 * object will attempt to read the corresponding dynamic symbol table.  In
 * order to bootstrap, the target uses the AT_BASE and AT_ENTRY aux vector
 * elements to locate the dynamic linker and executable mappings.  With these
 * mappings in place, we initialize a librtld_db agent on the target (see
 * mdb_pservice.c for how this is done), and then process each load object
 * found in the link-map chain.  In order to simplify the construction of
 * symbol tables for each load object, we would like make use of our existing
 * library of GElf processing code.  Since the MDB GElf code uses mdb_io
 * objects to read in an ELF file, we simply define a new type of mdb_io object
 * where each read operation is translated into a call to kproc's t_vread
 * function to read from the range of the address space defined by the mapping
 * as if it were a file.
```



```
 */

#include <sys/types.h>
#include <sys/proc.h>
#include <sys/auxv.h>

#include <strings.h>
#include <limits.h>
#include <rtld_db.h>
#include <procfs.h>
#include <dlfcn.h>
#include <kvm.h>

#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_string.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_ks.h>
#include <mdb/mdb_gelf.h>
#include <mdb/mdb_io_impl.h>
#include <mdb/mdb.h>

typedef struct kp_symarg {
        mdb_tgt_sym_f *sym_cb;          /* Caller's callback function */
        void *sym_data;                 /* Callback function argument */
        uint_t sym_type;                /* Symbol type/binding filter */
        uintptr_t sym_adjust;           /* Symbol value adjustment */
        mdb_syminfo_t sym_info;         /* Symbol id and table id */
        const char *sym_obj;            /* Containing object */
} kp_symarg_t;
______unchanged_portion_omitted_

static const mdb_tgt_ops_t kproc_ops = {
        (int (*)()) mdb_tgt_notsup,             /* t_setflags */
        kp_setcontext,                          /* t_setcontext */
        kp_activate,                            /* t_activate */
        kp_deactivate,                          /* t_deactivate */
        (void (*)()) mdb_tgt_nop,               /* t_periodic */
        kp_destroy,                             /* t_destroy */
        kp_name,                                /* t_name */
        kp_isa,                                 /* t_isa */
        kp_platform,                            /* t_platform */
        kp_uname,                               /* t_uname */
        kp_dmodel,                              /* t_dmodel */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_aread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_awrite */
        kp_vread,                               /* t_vread */
        kp_vwrite,                              /* t_vwrite */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_pread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_pwrite */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_fread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_fwrite */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_ioread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_iowrite */
        kp_vtop,                                /* t_vtop */
        kp_lookup_by_name,                      /* t_lookup_by_name */
        kp_lookup_by_addr,                      /* t_lookup_by_addr */
        kp_symbol_iter,                         /* t_symbol_iter */
        kp_mapping_iter,                        /* t_mapping_iter */
        kp_object_iter,                         /* t_object_iter */
        kp_addr_to_map,                         /* t_addr_to_map */
        kp_name_to_map,                         /* t_name_to_map */
        (struct ctf_file *(*)()) mdb_tgt_null,  /* t_addr_to_ctf */
        (struct ctf_file *(*)()) mdb_tgt_null,  /* t_name_to_ctf */
        kp_status,                              /* t_status */
        (int (*)()) mdb_tgt_notsup,             /* t_run */
```

```
 920         (int (*)()) mdb_tgt_notsup,             /* t_step */
 921         (int (*)()) mdb_tgt_notsup,             /* t_step_out */
 922         (int (*)()) mdb_tgt_notsup,             /* t_step_branch */
 922         (int (*)()) mdb_tgt_notsup,             /* t_next */
 923         (int (*)()) mdb_tgt_notsup,             /* t_cont */
 924         (int (*)()) mdb_tgt_notsup,             /* t_signal */
 925         (int (*)()) mdb_tgt_null,               /* t_add_sbrkpt */
 926         (int (*)()) mdb_tgt_null,               /* t_add_vbrkpt */
 927         (int (*)()) mdb_tgt_null,               /* t_add_pwapt */
 928         (int (*)()) mdb_tgt_null,               /* t_add_vwapt */
 929         (int (*)()) mdb_tgt_null,               /* t_add_iowapt */
 930         (int (*)()) mdb_tgt_null,               /* t_add_sysenter */
 931         (int (*)()) mdb_tgt_null,               /* t_add_sysexit */
 932         (int (*)()) mdb_tgt_null,               /* t_add_signal */
 933         (int (*)()) mdb_tgt_null,               /* t_add_fault */
 934         (int (*)()) mdb_tgt_notsup,             /* t_getareg XXX */
 935         (int (*)()) mdb_tgt_notsup,             /* t_putareg XXX */
 936         (int (*)()) mdb_tgt_notsup,             /* t_stack_iter XXX */
 937         kp_auxv                                 /* t_auxv */
 938 };
_____unchanged_portion_omitted_
```

```
**********************************************************
  148244 Fri Apr  6 17:24:54 2018
new/usr/src/cmd/mdb/common/mdb/mdb_proc.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2018 Joyent, Inc.
 * Copyright 2015 Joyent, Inc.
 * Copyright (c) 2014 by Delphix. All rights reserved.
 */

/*
 * User Process Target
 *
 * The user process target is invoked when the -u or -p command-line options
 * are used, or when an ELF executable file or ELF core file is specified on
 * the command-line.  This target is also selected by default when no target
 * options are present.  In this case, it defaults the executable name to
 * "a.out".  If no process or core file is currently attached, the target
 * functions as a kind of virtual /dev/zero (in accordance with adb(1)
 * semantics); reads from the virtual address space return zeroes and writes
 * fail silently.  The proc target itself is designed as a wrapper around the
 * services provided by libproc.so: t->t_pshandle is set to the struct
 * ps_prochandle pointer returned as a handle by libproc.  The target also
 * opens the executable file itself using the MDB GElf services, for
 * interpreting the .symtab and .dynsym if no libproc handle has been
 * initialized, and for handling i/o to and from the object file.  Currently,
 * the only ISA-dependent portions of the proc target are the $r and ::fpregs
 * dcmds, the callbacks for t_next() and t_step_out(), and the list of named
 * registers; these are linked in from the proc_isadep.c file for each ISA and
 * called from the common code in this file.
 *
 * The user process target implements complete user process control using the
 * facilities provided by libproc.so.  The MDB execution control model and
 * an overview of software event management is described in mdb_target.c.  The
 * proc target implements breakpoints by replacing the instruction of interest
 * with a trap instruction, and then restoring the original instruction to step
```



```
 * over the breakpoint.  The idea of replacing program text with instructions
 * that transfer control to the debugger dates back as far as 1951 [1].  When
 * the target stops, we replace each breakpoint with the original instruction
 * as part of the disarm operation.  This means that no special processing is
 * required for t_vread() because the instrumented instructions will never be
 * seen by the debugger once the target stops.  Some debuggers have improved
 * start/stop performance by leaving breakpoint traps in place and then
 * handling a read from a breakpoint address as a special case.  Although this
 * improves efficiency for a source-level debugger, it runs somewhat contrary
 * to the philosophy of the low-level debugger.  Since we remove the
 * instructions, users can apply other external debugging tools to the process
 * once it has stopped (e.g. the proc(1) tools) and not be misled by MDB
 * instrumentation.  The tracing of faults, signals, system calls, and
 * watchpoints and general process inspection is implemented directly using
 * the mechanisms provided by /proc, as described originally in [2] and [3].
 *
 * References
 *
 * [1] S. Gill, "The Diagnosis Of Mistakes In Programmes on the EDSAC",
 *     Proceedings of the Royal Society Series A Mathematical and Physical
 *     Sciences, Cambridge University Press, 206(1087), May 1951, pp. 538-554.
 *
 * [2] T.J. Killian, "Processes as Files", Proceedings of the USENIX Association
 *     Summer Conference, Salt Lake City, June 1984, pp. 203-207.
 *
 * [3] Roger Faulkner and Ron Gomes, "The Process File System and Process
 *     Model in UNIX System V", Proceedings of the USENIX Association
 *     Winter Conference, Dallas, January 1991, pp. 243-252.
 */

#include <mdb/mdb_proc.h>
#include <mdb/mdb_disasm.h>
#include <mdb/mdb_signal.h>
#include <mdb/mdb_string.h>
#include <mdb/mdb_module.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_conf.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_types.h>
#include <mdb/mdb.h>

#include <sys/utsname.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <termio.h>
#include <signal.h>
#include <stdio_ext.h>
#include <stdlib.h>
#include <string.h>

#define PC_FAKE         -1UL                    /* illegal pc value unequal 0 */
#define PANIC_BUFSIZE   1024

static const char PT_EXEC_PATH[] = "a.out";     /* Default executable */
static const char PT_CORE_PATH[] = "core";      /* Default core file */

static const pt_ptl_ops_t proc_lwp_ops;
static const pt_ptl_ops_t proc_tdb_ops;
static const mdb_se_ops_t proc_brkpt_ops;
static const mdb_se_ops_t proc_wapt_ops;

static int pt_setrun(mdb_tgt_t *, mdb_tgt_status_t *, int);
static void pt_activate_common(mdb_tgt_t *);
static mdb_tgt_vespec_f pt_ignore_sig;
static mdb_tgt_se_f pt_fork;
static mdb_tgt_se_f pt_exec;
```

```
 124 static int pt_lookup_by_name_thr(mdb_tgt_t *, const char *,
 125     const char *, GElf_Sym *, mdb_syminfo_t *, mdb_tgt_tid_t);
 126 static int tlsbase(mdb_tgt_t *, mdb_tgt_tid_t, Lmid_t, const char *,
 127     psaddr_t *);

 129 /*
 130  * When debugging postmortem, we don't resolve names as we may very well not
 131  * be on a system on which those names resolve.
 132  */
 133 #define PT_LIBPROC_RESOLVE(P) \
 134         (!(mdb.m_flags & MDB_FL_LMRAW) && Pstate(P) != PS_DEAD)

 136 /*
 137  * The Perror_printf() function interposes on the default, empty libproc
 138  * definition.  It will be called to report additional information on complex
 139  * errors, such as a corrupt core file.  We just pass the args to vwarn.
 140  */
 141 /*ARGSUSED*/
 142 void
 143 Perror_printf(struct ps_prochandle *P, const char *format, ...)
 144 {
 145         va_list alist;

 147         va_start(alist, format);
 148         vwarn(format, alist);
 149         va_end(alist);
 150 }
______unchanged_portion_omitted_


4662 static const mdb_tgt_ops_t proc_ops = {
4663         pt_setflags,                            /* t_setflags */
4664         (int (*)()) mdb_tgt_notsup,             /* t_setcontext */
4665         pt_activate,                            /* t_activate */
4666         pt_deactivate,                          /* t_deactivate */
4667         pt_periodic,                            /* t_periodic */
4668         pt_destroy,                             /* t_destroy */
4669         pt_name,                                /* t_name */
4670         (const char *(*)()) mdb_conf_isa,       /* t_isa */
4671         pt_platform,                            /* t_platform */
4672         pt_uname,                               /* t_uname */
4673         pt_dmodel,                              /* t_dmodel */
4674         (ssize_t (*)()) mdb_tgt_notsup,         /* t_aread */
4675         (ssize_t (*)()) mdb_tgt_notsup,         /* t_awrite */
4676         pt_vread,                               /* t_vread */
4677         pt_vwrite,                              /* t_vwrite */
4678         (ssize_t (*)()) mdb_tgt_notsup,         /* t_pread */
4679         (ssize_t (*)()) mdb_tgt_notsup,         /* t_pwrite */
4680         pt_fread,                               /* t_fread */
4681         pt_fwrite,                              /* t_fwrite */
4682         (ssize_t (*)()) mdb_tgt_notsup,         /* t_ioread */
4683         (ssize_t (*)()) mdb_tgt_notsup,         /* t_iowrite */
4684         (int (*)()) mdb_tgt_notsup,             /* t_vtop */
4685         pt_lookup_by_name,                      /* t_lookup_by_name */
4686         pt_lookup_by_addr,                      /* t_lookup_by_addr */
4687         pt_symbol_iter,                         /* t_symbol_iter */
4688         pt_mapping_iter,                        /* t_mapping_iter */
4689         pt_object_iter,                         /* t_object_iter */
4690         pt_addr_to_map,                         /* t_addr_to_map */
4691         pt_name_to_map,                         /* t_name_to_map */
4692         pt_addr_to_ctf,                         /* t_addr_to_ctf */
4693         pt_name_to_ctf,                         /* t_name_to_ctf */
4694         pt_status,                              /* t_status */
4695         pt_run,                                 /* t_run */
4696         pt_step,                                /* t_step */
```



```
4697         pt_step_out,                           /* t_step_out */
4698         (int (*)()) mdb_tgt_notsup,            /* t_step_branch */
4698         pt_next,                               /* t_next */
4699         pt_continue,                           /* t_cont */
4700         pt_signal,                             /* t_signal */
4701         pt_add_vbrkpt,                         /* t_add_vbrkpt */
4702         pt_add_sbrkpt,                         /* t_add_sbrkpt */
4703         (int (*)()) mdb_tgt_null,              /* t_add_pwapt */
4704         pt_add_vwapt,                          /* t_add_vwapt */
4705         (int (*)()) mdb_tgt_null,              /* t_add_iowapt */
4706         pt_add_sysenter,                       /* t_add_sysenter */
4707         pt_add_sysexit,                        /* t_add_sysexit */
4708         pt_add_signal,                         /* t_add_signal */
4709         pt_add_fault,                          /* t_add_fault */
4710         pt_getareg,                            /* t_getareg */
4711         pt_putareg,                            /* t_putareg */
4712         pt_stack_iter,                         /* t_stack_iter */
4713         pt_auxv                                /* t_auxv */
4714 };
______unchanged_portion_omitted_
```

```
**********************************************************
   11832 Fri Apr  6 17:24:54 2018
new/usr/src/cmd/mdb/common/mdb/mdb_rawfile.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

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

/*
 * Raw File Target
 *
 * The raw file target is invoked whenever a file of unrecognizable type is
 * specified on the command line, or when raw file examination is forced using
 * the -f option.  If one file is specified, that file will be opened as the
 * "object" file.  If two files are specified, the second one will be opened
 * as the "core" file.  Each file is opened using the fdio backend, which
 * internally supports both byte-oriented i/o and block-oriented i/o as needed.
 */

#include <mdb/mdb_modapi.h>
#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_io_impl.h>
#include <mdb/mdb_conf.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb.h>

#include <sys/dtrace.h>
#include <fcntl.h>

typedef struct rf_data {
        mdb_io_t *r_object_fio;
        mdb_io_t *r_core_fio;
} rf_data_t;
```

*unchanged_portion_omitted_*

```
static const mdb_tgt_ops_t rawfile_ops = {
```



```
        rf_setflags,                            /* t_setflags */
        (int (*)()) mdb_tgt_notsup,             /* t_setcontext */
        rf_activate,                            /* t_activate */
        rf_deactivate,                          /* t_deactivate */
        (void (*)()) mdb_tgt_nop,               /* t_periodic */
        rf_destroy,                             /* t_destroy */
        rf_name,                                /* t_name */
        (const char *(*)()) mdb_conf_isa,       /* t_isa */
        (const char *(*)()) mdb_conf_platform,  /* t_platform */
        (int (*)()) mdb_tgt_notsup,             /* t_uname */
        (int (*)()) mdb_tgt_notsup,             /* t_dmodel */
        rf_aread,                               /* t_aread */
        rf_awrite,                              /* t_awrite */
        rf_vread,                               /* t_vread */
        rf_vwrite,                              /* t_vwrite */
        rf_pread,                               /* t_pread */
        rf_pwrite,                              /* t_pwrite */
        rf_fread,                               /* t_fread */
        rf_fwrite,                              /* t_fwrite */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_ioread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_iowrite */
        (int (*)()) mdb_tgt_notsup,             /* t_vtop */
        (int (*)()) mdb_tgt_notsup,             /* t_lookup_by_name */
        (int (*)()) mdb_tgt_notsup,             /* t_lookup_by_addr */
        (int (*)()) mdb_tgt_notsup,             /* t_symbol_iter */
        rf_mapping_iter,                        /* t_mapping_iter */
        rf_mapping_iter,                        /* t_object_iter */
        (const mdb_map_t *(*)()) mdb_tgt_null,  /* t_addr_to_map */
        (const mdb_map_t *(*)()) mdb_tgt_null,  /* t_name_to_map */
        (struct ctf_file *(*)()) mdb_tgt_null,  /* t_addr_to_ctf */
        (struct ctf_file *(*)()) mdb_tgt_null,  /* t_name_to_ctf */
        rf_status,                              /* t_status */
        (int (*)()) mdb_tgt_notsup,             /* t_run */
        (int (*)()) mdb_tgt_notsup,             /* t_step */
        (int (*)()) mdb_tgt_notsup,             /* t_step_out */
        (int (*)()) mdb_tgt_notsup,             /* t_step_branch */
        (int (*)()) mdb_tgt_notsup,             /* t_next */
        (int (*)()) mdb_tgt_notsup,             /* t_cont */
        (int (*)()) mdb_tgt_notsup,             /* t_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_vbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_sbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_pwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_vwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_iowapt */
        (int (*)()) mdb_tgt_null,               /* t_add_sysenter */
        (int (*)()) mdb_tgt_null,               /* t_add_sysexit */
        (int (*)()) mdb_tgt_null,               /* t_add_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_fault */
        (int (*)()) mdb_tgt_notsup,             /* t_getareg */
        (int (*)()) mdb_tgt_notsup,             /* t_putareg */
        (int (*)()) mdb_tgt_notsup,             /* t_stack_iter */
        (int (*)()) mdb_tgt_notsup              /* t_auxv */
};
```

*unchanged_portion_omitted_*

```
**********************************************************
   64895 Fri Apr  6 17:24:54 2018
new/usr/src/cmd/mdb/common/mdb/mdb_target.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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.
 */

/*
 * MDB Target Layer
 *
 * The *target* is the program being inspected by the debugger.  The MDB target
 * layer provides a set of functions that insulate common debugger code,
 * including the MDB Module API, from the implementation details of how the
 * debugger accesses information from a given target.  Each target exports a
 * standard set of properties, including one or more address  spaces, one or
 * more symbol tables, a set of load objects, and a set of threads that can be
 * examined using the interfaces in <mdb/mdb_target.h>.  This technique has
 * been employed successfully in other debuggers, including [1], primarily
 * to improve portability, although the term "target" often refers to the
 * encapsulation of architectural or operating system-specific details.  The
 * target abstraction is useful for MDB because it allows us to easily extend
 * the debugger to examine a variety of different program forms.  Primarily,
 * the target functions validate input arguments and then call an appropriate
 * function in the target ops vector, defined in <mdb/mdb_target_impl.h>.
 * However, this interface layer provides a very high level of flexibility for
 * separating the debugger interface from instrumentation details.  Experience
 * has shown this kind of design can facilitate separating out debugger
 * instrumentation into an external agent [2] and enable the development of
 * advanced instrumentation frameworks [3].  We want MDB to be an ideal
 * extensible framework for the development of such applications.
 *
 * Aside from a set of wrapper functions, the target layer also provides event
 * management for targets that represent live executing programs.  Our model of
 * events is also extensible, and is based upon work in [3] and [4].  We define
 * a *software event* as a state transition in the target program (for example,
 * the transition of the program counter to a location of interest) that is
 * observed by the debugger or its agent.  A *software event specifier* is a
```



```
 * description of a class of software events that is used by the debugger to
 * instrument the target so that the corresponding software events can be
 * observed.  In MDB, software event specifiers are represented by the
 * mdb_sespec_t structure, defined in <mdb/mdb_target_impl.h>.  As the user,
 * the internal debugger code, and MDB modules may all wish to observe software
 * events and receive appropriate notification and callbacks, we do not expose
 * software event specifiers directly as part of the user interface.  Instead,
 * clients of the target layer request that events be observed by creating
 * new *virtual event specifiers*.  Each virtual specifier is named by a unique
 * non-zero integer (the VID), and is represented by a mdb_vespec_t structure.
 * One or more virtual specifiers are then associated with each underlying
 * software event specifier.  This design enforces the constraint that the
 * target must only insert one set of instrumentation, regardless of how many
 * times the target layer was asked to trace a given event.  For example, if
 * multiple clients request a breakpoint at a particular address, the virtual
 * specifiers will map to the same sespec, ensuring that only one breakpoint
 * trap instruction is actually planted at the given target address.  When no
 * virtual specifiers refer to an sespec, it is no longer needed and can be
 * removed, along with the corresponding instrumentation.
 *
 * The following state transition diagram illustrates the life cycle of a
 * software event specifier and example transitions:
 *
 *                                        cont/
 *     +--------+   delete   +--------+    stop    +-------+
 *    (|( DEAD )|) <------- (  ACTIVE  ) <------> (  ARMED  )
 *     +--------+            +--------+           +-------+
 *          ^   load/unload  ^        ^   failure/     |
 *   delete |        object /          \  reset        | failure
 *          |              v            v             |
 *          |      +--------+          +-------+       |
 *          +---- (   IDLE   )        (   ERR   ) <----+
 *          |      +--------+          +-------+
 *          |                             |
 *          +-----------------------------+
 *
 * The MDB execution control model is based upon the synchronous debugging
 * model exported by Solaris proc(4).  A target program is set running or the
 * debugger is attached to a running target.  On ISTOP (stop on event of
 * interest), one target thread is selected as the representative.  The
 * algorithm for selecting the representative is target-specific, but we assume
 * that if an observed software event has occurred, the target will select the
 * thread that triggered the state transition of interest.  The other threads
 * are stopped in sympathy with the representative as soon as possible.  Prior
 * to continuing the target, we plant our instrumentation, transitioning event
 * specifiers from the ACTIVE to the ARMED state, and then back again when the
 * target stops.  We then query each active event specifier to learn which ones
 * are matched, and then invoke the callbacks associated with their vespecs.
 * If an OS error occurs while attempting to arm or disarm a specifier, the
 * specifier is transitioned to the ERROR state; we will attempt to arm it
 * again at the next continue.  If no target process is under our control or
 * if an event is not currently applicable (e.g. a deferred breakpoint on an
 * object that is not yet loaded), it remains in the IDLE state.  The target
 * implementation should intercept object load events and then transition the
 * specifier to the ACTIVE state when the corresponding object is loaded.
 *
 * To simplify the debugger implementation and allow targets to easily provide
 * new types of observable events, most of the event specifier management is
 * done by the target layer.  Each software event specifier provides an ops
 * vector of subroutines that the target layer can call to perform the
 * various state transitions described above.  The target maintains two lists
 * of mdb_sespec_t's: the t_idle list (IDLE state) and the t_active list
 * (ACTIVE, ARMED, and ERROR states).  Each mdb_sespec_t maintains a list of
 * associated mdb_vespec_t's.  If an sespec is IDLE or ERROR, its se_errno
 * field will have an errno value specifying the reason for its inactivity.
 * The vespec stores the client's callback function and private data, and the
```

```
 124  * arguments used to construct the sespec.  All objects are reference counted
 125  * so we can destroy an object when it is no longer needed.  The mdb_sespec_t
 126  * invariants for the respective states are as follows:
 127  *
 128  *   IDLE: on t_idle list, se_data == NULL, se_errno != 0, se_ctor not called
 129  * ACTIVE: on t_active list, se_data valid, se_errno == 0, se_ctor called
 130  *  ARMED: on t_active list, se_data valid, se_errno == 0, se_ctor called
 131  *  ERROR: on t_active list, se_data valid, se_errno != 0, se_ctor called
 132  *
 133  * Additional commentary on specific state transitions and issues involving
 134  * event management can be found below near the target layer functions.
 135  *
 136  * References
 137  *
 138  * [1] John Gilmore, "Working in GDB", Technical Report, Cygnus Support,
 139  *     1.84 edition, 1994.
 140  *
 141  * [2] David R. Hanson and Mukund Raghavachari, "A Machine-Independent
 142  *     Debugger", Software--Practice and Experience, 26(11), 1277-1299(1996).
 143  *
 144  * [3] Michael W. Shapiro, "RDB: A System for Incremental Replay Debugging",
 145  *     Technical Report CS-97-12, Department of Computer Science,
 146  *     Brown University.
 147  *
 148  * [4] Daniel B. Price, "New Techniques for Replay Debugging", Technical
 149  *     Report CS-98-05, Department of Computer Science, Brown University.
 150  */

 152 #include <mdb/mdb_target_impl.h>
 153 #include <mdb/mdb_debug.h>
 154 #include <mdb/mdb_modapi.h>
 155 #include <mdb/mdb_err.h>
 156 #include <mdb/mdb_callb.h>
 157 #include <mdb/mdb_gelf.h>
 158 #include <mdb/mdb_io_impl.h>
 159 #include <mdb/mdb_string.h>
 160 #include <mdb/mdb_signal.h>
 161 #include <mdb/mdb_frame.h>
 162 #include <mdb/mdb.h>

 164 #include <sys/stat.h>
 165 #include <sys/param.h>
 166 #include <sys/signal.h>
 167 #include <strings.h>
 168 #include <stdlib.h>
 169 #include <errno.h>

 171 /*
 172  * Define convenience macros for referencing the set of vespec flag bits that
 173  * are preserved by the target implementation, and the set of bits that
 174  * determine automatic ve_hits == ve_limit behavior.
 175  */
 176 #define T_IMPL_BITS     \
 177         (MDB_TGT_SPEC_INTERNAL | MDB_TGT_SPEC_SILENT | MDB_TGT_SPEC_MATCHED | \
 178         MDB_TGT_SPEC_DELETED)

 180 #define T_AUTO_BITS     \
 181         (MDB_TGT_SPEC_AUTOSTOP | MDB_TGT_SPEC_AUTODEL | MDB_TGT_SPEC_AUTODIS)

 183 /*
 184  * Define convenience macro for referencing target flag pending continue bits.
 185  */
 186 #define T_CONT_BITS     \
 187         (MDB_TGT_F_STEP | MDB_TGT_F_STEP_OUT | MDB_TGT_F_NEXT | MDB_TGT_F_CONT)
 185         (MDB_TGT_F_STEP | MDB_TGT_F_STEP_OUT | MDB_TGT_F_STEP_BRANCH | \
 186         MDB_TGT_F_NEXT | MDB_TGT_F_CONT)
```



```
 189 mdb_tgt_t *
 190 mdb_tgt_create(mdb_tgt_ctor_f *ctor, int flags, int argc, const char *argv[])
 191 {
 192         mdb_module_t *mp;
 193         mdb_tgt_t *t;

 195         if (flags & ~MDB_TGT_F_ALL) {
 196                 (void) set_errno(EINVAL);
 197                 return (NULL);
 198         }

 200         t = mdb_zalloc(sizeof (mdb_tgt_t), UM_SLEEP);
 201         mdb_list_append(&mdb.m_tgtlist, t);

 203         t->t_module = &mdb.m_rmod;
 204         t->t_matched = T_SE_END;
 205         t->t_flags = flags;
 206         t->t_vepos = 1;
 207         t->t_veneg = 1;

 209         for (mp = mdb.m_mhead; mp != NULL; mp = mp->mod_next) {
 210                 if (ctor == mp->mod_tgt_ctor) {
 211                         t->t_module = mp;
 212                         break;
 213                 }
 214         }

 216         if (ctor(t, argc, argv) != 0) {
 217                 mdb_list_delete(&mdb.m_tgtlist, t);
 218                 mdb_free(t, sizeof (mdb_tgt_t));
 219                 return (NULL);
 220         }

 222         mdb_dprintf(MDB_DBG_TGT, "t_create %s (%p)\n",
 223             t->t_module->mod_name, (void *)t);

 225         (void) t->t_ops->t_status(t, &t->t_status);
 226         return (t);
 227 }
_____unchanged_portion_omitted_

1047 /*
1048  * This function provides the low-level target continue algorithm.  We proceed
1049  * in three phases: (1) we arm the active sespecs, except the specs matched at
1050  * the time we last stopped, (2) we call se_cont() on any matched sespecs to
1051  * step over these event transitions, and then arm the corresponding sespecs,
1052  * and (3) we call the appropriate low-level continue routine.  Once the
1053  * target stops again, we determine which sespecs were matched, and invoke the
1054  * appropriate vespec callbacks and perform other vespec maintenance.
1055  */
1056 static int
1057 tgt_continue(mdb_tgt_t *t, mdb_tgt_status_t *tsp,
1058     int (*t_cont)(mdb_tgt_t *, mdb_tgt_status_t *))
1059 {
1060         mdb_var_t *hitv = mdb_nv_lookup(&mdb.m_nv, "hits");
1061         uintptr_t pc = t->t_status.st_pc;
1062         int error = 0;

1064         mdb_sespec_t *sep, *nsep, *matched;
1065         mdb_vespec_t *vep, *nvep;
1066         uintptr_t addr;

1068         uint_t cbits = 0;       /* union of pending continue bits */
1069         uint_t ncont = 0;       /* # of callbacks that requested cont */
1070         uint_t n = 0;           /* # of callbacks */
```

```
1072         /*
1073          * If the target is undead, dead, or lost, we no longer allow continue.
1074          * This effectively forces the user to use ::kill or ::run after death.
1075          */
1076         if (t->t_status.st_state == MDB_TGT_UNDEAD)
1077                 return (set_errno(EMDB_TGTZOMB));
1078         if (t->t_status.st_state == MDB_TGT_DEAD)
1079                 return (set_errno(EMDB_TGTCORE));
1080         if (t->t_status.st_state == MDB_TGT_LOST)
1081                 return (set_errno(EMDB_TGTLOST));

1083         /*
1084          * If any of single-step, step-over, or step-out is pending, it takes
1085          * precedence over an explicit or pending continue, because these are
1086          * all different specialized forms of continue.
1087          */
1088         if (t->t_flags & MDB_TGT_F_STEP)
1089                 t_cont = t->t_ops->t_step;
1090         else if (t->t_flags & MDB_TGT_F_NEXT)
1091                 t_cont = t->t_ops->t_step;
1091         else if (t->t_flags & MDB_TGT_F_STEP_BRANCH)
1092                 t_cont = t->t_ops->t_cont;
1092         else if (t->t_flags & MDB_TGT_F_STEP_OUT)
1093                 t_cont = t->t_ops->t_cont;

1095         /*
1096          * To handle step-over, we ask the target to find the address past the
1097          * next control transfer instruction.  If an address is found, we plant
1098          * a temporary breakpoint there and continue; otherwise just step.
1099          */
1100         if ((t->t_flags & MDB_TGT_F_NEXT) && !(t->t_flags & MDB_TGT_F_STEP)) {
1101                 if (t->t_ops->t_next(t, &addr) == -1 || mdb_tgt_add_vbrkpt(t,
1102                     addr, MDB_TGT_SPEC_HIDDEN | MDB_TGT_SPEC_TEMPORARY,
1103                     no_se_f, NULL) == 0) {
1104                         mdb_dprintf(MDB_DBG_TGT, "next falling back to step: "
1105                             "%s\n", mdb_strerror(errno));
1106                 } else
1107                         t_cont = t->t_ops->t_cont;
1108         }

1110         /*
1111          * To handle step-out, we ask the target to find the return address of
1112          * the current frame, plant a temporary breakpoint there, and continue.
1113          */
1114         if (t->t_flags & MDB_TGT_F_STEP_OUT) {
1115                 if (t->t_ops->t_step_out(t, &addr) == -1)
1116                         return (-1); /* errno is set for us */

1118                 if (mdb_tgt_add_vbrkpt(t, addr, MDB_TGT_SPEC_HIDDEN |
1119                     MDB_TGT_SPEC_TEMPORARY, no_se_f, NULL) == 0)
1120                         return (-1); /* errno is set for us */
1121         }

1124         /*
1125          * To handle step-branch, we ask the target to enable it for the coming
1126          * continue.  Step-branch is incompatible with step, so don't enable it
1127          * if we're going to be stepping.
1128          */
1129         if (t->t_flags & MDB_TGT_F_STEP_BRANCH && t_cont == t->t_ops->t_cont) {
1130                 if (t->t_ops->t_step_branch(t) == -1)
1131                         return (-1); /* errno is set for us */
1132         }

1123         (void) mdb_signal_block(SIGHUP);
1124         (void) mdb_signal_block(SIGTERM);
```



```
1125         mdb_intr_disable();

1127         t->t_flags &= ~T_CONT_BITS;
1128         t->t_flags |= MDB_TGT_F_BUSY;
1129         mdb_tgt_sespec_arm_all(t);

1131         ASSERT(t->t_matched != NULL);
1132         matched = t->t_matched;
1133         t->t_matched = T_SE_END;

1135         if (mdb.m_term != NULL)
1136                 IOP_SUSPEND(mdb.m_term);

1138         /*
1139          * Iterate over the matched sespec list, performing autostop processing
1140          * and clearing the matched bit for each associated vespec.  We then
1141          * invoke each sespec's se_cont callback in order to continue past
1142          * the corresponding event.  If the matched list has more than one
1143          * sespec, we assume that the se_cont callbacks are non-interfering.
1144          */
1145         for (sep = matched; sep != T_SE_END; sep = sep->se_matched) {
1146                 for (vep = mdb_list_next(&sep->se_velist); vep != NULL; ) {
1147                         if ((vep->ve_flags & MDB_TGT_SPEC_AUTOSTOP) &&
1148                             (vep->ve_limit && vep->ve_hits == vep->ve_limit))
1149                                 vep->ve_hits = 0;

1151                         vep->ve_flags &= ~MDB_TGT_SPEC_MATCHED;
1152                         vep = mdb_list_next(vep);
1153                 }

1155                 if (sep->se_ops->se_cont(t, sep, &t->t_status) == -1) {
1156                         error = errno ? errno : -1;
1157                         tgt_disarm_sespecs(t);
1158                         break;
1159                 }

1161                 if (!(t->t_status.st_flags & MDB_TGT_ISTOP)) {
1162                         tgt_disarm_sespecs(t);
1163                         if (t->t_status.st_state == MDB_TGT_UNDEAD)
1164                                 mdb_tgt_sespec_idle_all(t, EMDB_TGTZOMB, TRUE);
1165                         else if (t->t_status.st_state == MDB_TGT_LOST)
1166                                 mdb_tgt_sespec_idle_all(t, EMDB_TGTLOST, TRUE);
1167                         break;
1168                 }
1169         }

1171         /*
1172          * Clear the se_matched field for each matched sespec, and drop the
1173          * reference count since the sespec is no longer on the matched list.
1174          */
1175         for (sep = matched; sep != T_SE_END; sep = nsep) {
1176                 nsep = sep->se_matched;
1177                 sep->se_matched = NULL;
1178                 mdb_tgt_sespec_rele(t, sep);
1179         }

1181         /*
1182          * If the matched list was non-empty, see if we hit another event while
1183          * performing se_cont() processing.  If so, don't bother continuing any
1184          * further.  If not, arm the sespecs on the old matched list by calling
1185          * mdb_tgt_sespec_arm_all() again and then continue by calling t_cont.
1186          */
1187         if (matched != T_SE_END) {
1188                 if (error != 0 || !(t->t_status.st_flags & MDB_TGT_ISTOP))
1189                         goto out; /* abort now if se_cont() failed */
```

```
1191                 if ((t->t_matched = tgt_match_sespecs(t, FALSE)) != T_SE_END) {
1192                         tgt_disarm_sespecs(t);
1193                         goto out;
1194                 }

1196                 mdb_tgt_sespec_arm_all(t);
1197         }

1199         if (t_cont != t->t_ops->t_step || pc == t->t_status.st_pc) {
1200                 if (t_cont(t, &t->t_status) != 0)
1201                         error = errno ? errno : -1;
1202         }

1204         tgt_disarm_sespecs(t);

1206         if (t->t_flags & MDB_TGT_F_UNLOAD)
1207                 longjmp(mdb.m_frame->f_pcb, MDB_ERR_QUIT);

1209         if (t->t_status.st_state == MDB_TGT_UNDEAD)
1210                 mdb_tgt_sespec_idle_all(t, EMDB_TGTZOMB, TRUE);
1211         else if (t->t_status.st_state == MDB_TGT_LOST)
1212                 mdb_tgt_sespec_idle_all(t, EMDB_TGTLOST, TRUE);
1213         else if (t->t_status.st_flags & MDB_TGT_ISTOP)
1214                 t->t_matched = tgt_match_sespecs(t, TRUE);
1215 out:
1216         if (mdb.m_term != NULL)
1217                 IOP_RESUME(mdb.m_term);

1219         (void) mdb_signal_unblock(SIGTERM);
1220         (void) mdb_signal_unblock(SIGHUP);
1221         mdb_intr_enable();

1223         for (sep = t->t_matched; sep != T_SE_END; sep = sep->se_matched) {
1224                 /*
1225                  * When we invoke a ve_callback, it may in turn request that the
1226                  * target continue immediately after callback processing is
1227                  * complete.  We only allow this to occur if *all* callbacks
1228                  * agree to continue.  To implement this behavior, we keep a
1229                  * count (ncont) of such requests, and only apply the cumulative
1230                  * continue bits (cbits) to the target if ncont is equal to the
1231                  * total number of callbacks that are invoked (n).
1232                  */
1233                 for (vep = mdb_list_next(&sep->se_velist);
1234                     vep != NULL; vep = nvep, n++) {
1235                         /*
1236                          * Place an extra hold on the current vespec and pick
1237                          * up the next pointer before invoking the callback: we
1238                          * must be prepared for the vespec to be deleted or
1239                          * moved to a different list by the callback.
1240                          */
1241                         mdb_tgt_vespec_hold(t, vep);
1242                         nvep = mdb_list_next(vep);

1244                         vep->ve_flags |= MDB_TGT_SPEC_MATCHED;
1245                         vep->ve_hits++;

1247                         mdb_nv_set_value(mdb.m_dot, t->t_status.st_pc);
1248                         mdb_nv_set_value(hitv, vep->ve_hits);

1250                         ASSERT((t->t_flags & T_CONT_BITS) == 0);
1251                         vep->ve_callback(t, vep->ve_id, vep->ve_data);

1253                         ncont += (t->t_flags & T_CONT_BITS) != 0;
1254                         cbits |= (t->t_flags & T_CONT_BITS);
1255                         t->t_flags &= ~T_CONT_BITS;
```



```
1257                         if (vep->ve_limit && vep->ve_hits == vep->ve_limit) {
1258                                 if (vep->ve_flags & MDB_TGT_SPEC_AUTODEL)
1259                                         (void) mdb_tgt_vespec_delete(t,
1260                                             vep->ve_id);
1261                                 else if (vep->ve_flags & MDB_TGT_SPEC_AUTODIS)
1262                                         (void) mdb_tgt_vespec_disable(t,
1263                                             vep->ve_id);
1264                         }

1266                         if (vep->ve_limit && vep->ve_hits < vep->ve_limit) {
1267                                 if (vep->ve_flags & MDB_TGT_SPEC_AUTOSTOP)
1268                                         (void) mdb_tgt_continue(t, NULL);
1269                         }

1271                         mdb_tgt_vespec_rele(t, vep);
1272                 }
1273         }

1275         if (t->t_matched != T_SE_END && ncont == n)
1276                 t->t_flags |= cbits; /* apply continues (see above) */

1278         mdb_tgt_sespec_prune_all(t);

1280         t->t_status.st_flags &= ~MDB_TGT_BUSY;
1281         t->t_flags &= ~MDB_TGT_F_BUSY;

1283         if (tsp != NULL)
1284                 bcopy(&t->t_status, tsp, sizeof (mdb_tgt_status_t));

1286         if (error != 0)
1287                 return (set_errno(error));

1289         return (0);
1290 }
______unchanged_portion_omitted_

1398 int
1399 mdb_tgt_step_branch(mdb_tgt_t *t, mdb_tgt_status_t *tsp)
1400 {
1401         t->t_flags |= MDB_TGT_F_STEP_BRANCH; /* set flag even if tgt not busy */
1402         return (tgt_request_continue(t, tsp, 0, t->t_ops->t_cont));
1403 }

1387 int
1388 mdb_tgt_next(mdb_tgt_t *t, mdb_tgt_status_t *tsp)
1389 {
1390         t->t_flags |= MDB_TGT_F_NEXT; /* set flag even if tgt not busy */
1391         return (tgt_request_continue(t, tsp, 0, t->t_ops->t_step));
1392 }
______unchanged_portion_omitted_
```

```
**********************************************************
   22252 Fri Apr  6 17:24:55 2018
new/usr/src/cmd/mdb/common/mdb/mdb_target.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (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 2005 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

#ifndef _MDB_TARGET_H
#define _MDB_TARGET_H

#include <sys/utsname.h>
#include <sys/types.h>
#include <gelf.h>

#ifdef __cplusplus
extern "C" {
#endif

/*
 * Forward declaration of the target structure: the target itself is defined in
 * mdb_tgt_impl.h and is opaque with respect to callers of this interface.
 */

struct mdb_tgt;
struct mdb_arg;
struct ctf_file;

typedef struct mdb_tgt mdb_tgt_t;

extern void mdb_create_builtin_tgts(void);
extern void mdb_create_loadable_disasms(void);

/*
 * Target Constructors
 *
 * These functions are used to create a complete debugger target.  The
```



```
 * constructor is passed as an argument to mdb_tgt_create().
 */

extern int mdb_value_tgt_create(mdb_tgt_t *, int, const char *[]);
#ifndef _KMDB
extern int mdb_kvm_tgt_create(mdb_tgt_t *, int, const char *[]);
extern int mdb_proc_tgt_create(mdb_tgt_t *, int, const char *[]);
extern int mdb_kproc_tgt_create(mdb_tgt_t *, int, const char *[]);
extern int mdb_rawfile_tgt_create(mdb_tgt_t *, int, const char *[]);
#else
extern int kmdb_kvm_create(mdb_tgt_t *, int, const char *[]);
#endif

/*
 * Targets are created by calling mdb_tgt_create() with an optional set of
 * target flags, an argument list, and a target constructor (see above):
 */

#define MDB_TGT_F_RDWR          0x0001  /* Open for writing (else read-only) */
#define MDB_TGT_F_ALLOWIO       0x0002  /* Allow I/O mem access (live only) */
#define MDB_TGT_F_FORCE         0x0004  /* Force open (even if non-exclusive) */
#define MDB_TGT_F_PRELOAD       0x0008  /* Preload all symbol tables */
#define MDB_TGT_F_NOLOAD        0x0010  /* Do not do load-object processing */
#define MDB_TGT_F_NOSTOP        0x0020  /* Do not stop target on attach */
#define MDB_TGT_F_STEP          0x0040  /* Single-step is pending */
#define MDB_TGT_F_STEP_OUT      0x0080  /* Step-out is pending */
#define MDB_TGT_F_NEXT          0x0100  /* Step-over is pending */
#define MDB_TGT_F_CONT          0x0200  /* Continue is pending */
#define MDB_TGT_F_BUSY          0x0400  /* Target is busy executing */
#define MDB_TGT_F_ASIO          0x0800  /* Use t_aread and t_awrite for i/o */
#define MDB_TGT_F_UNLOAD        0x1000  /* Unload has been requested */
#define MDB_TGT_F_ALL           0x1fff  /* Mask of all valid flags */
#define MDB_TGT_F_STEP_BRANCH   0x0100  /* Step-branch is pending */
#define MDB_TGT_F_NEXT          0x0200  /* Step-over is pending */
#define MDB_TGT_F_CONT          0x0400  /* Continue is pending */
#define MDB_TGT_F_BUSY          0x0800  /* Target is busy executing */
#define MDB_TGT_F_ASIO          0x1000  /* Use t_aread and t_awrite for i/o */
#define MDB_TGT_F_UNLOAD        0x2000  /* Unload has been requested */
#define MDB_TGT_F_ALL           0x3fff  /* Mask of all valid flags */

typedef int mdb_tgt_ctor_f(mdb_tgt_t *, int, const char *[]);

extern mdb_tgt_t *mdb_tgt_create(mdb_tgt_ctor_f *, int, int, const char *[]);
extern void mdb_tgt_destroy(mdb_tgt_t *);

extern int mdb_tgt_getflags(mdb_tgt_t *);
extern int mdb_tgt_setflags(mdb_tgt_t *, int);
extern int mdb_tgt_setcontext(mdb_tgt_t *, void *);

/*
 * Targets are activated and de-activated by the debugger framework.  An
 * activation occurs after construction when the target becomes the current
 * target in the debugger.  A target is de-activated prior to its destructor
 * being called by mdb_tgt_destroy, or when another target is activated.
 * These callbacks are suitable for loading support modules and other tasks.
 */
extern void mdb_tgt_activate(mdb_tgt_t *);

/*
 * Prior to issuing a new command prompt, the debugger framework calls the
 * target's periodic callback to allow it to load new modules or perform
 * other background tasks.
 */
extern void mdb_tgt_periodic(mdb_tgt_t *);

/*
```

```
 117  * Convenience functions for accessing miscellaneous target information.
 118  */
 119 extern const char *mdb_tgt_name(mdb_tgt_t *);
 120 extern const char *mdb_tgt_isa(mdb_tgt_t *);
 121 extern const char *mdb_tgt_platform(mdb_tgt_t *);
 122 extern int mdb_tgt_uname(mdb_tgt_t *, struct utsname *);
 123 extern int mdb_tgt_dmodel(mdb_tgt_t *);

 125 /*
 126  * Address Space Interface
 127  *
 128  * Each target can provide access to a set of address spaces, which may include
 129  * a primary virtual address space, a physical address space, an object file
 130  * address space (where virtual addresses are converted to file offsets in an
 131  * object file), and an I/O port address space.  Additionally, the target can
 132  * provide access to alternate address spaces, which are identified by the
 133  * opaque mdb_tgt_as_t type.  If the 'as' parameter to mdb_tgt_aread or
 134  * mdb_tgt_awrite is one of the listed constants, these calls are equivalent
 135  * to mdb_tgt_{v|p|f|io}read or write.
 136  */

 138 typedef void *          mdb_tgt_as_t;           /* Opaque address space id */
 139 typedef uint64_t        mdb_tgt_addr_t;         /* Generic unsigned address */
 140 typedef uint64_t        physaddr_t;             /* Physical memory address */

 142 #define MDB_TGT_AS_VIRT ((mdb_tgt_as_t)-1L)     /* Virtual address space */
 143 #define MDB_TGT_AS_PHYS ((mdb_tgt_as_t)-2L)     /* Physical address space */
 144 #define MDB_TGT_AS_FILE ((mdb_tgt_as_t)-3L)     /* Object file address space */
 145 #define MDB_TGT_AS_IO   ((mdb_tgt_as_t)-4L)     /* I/o address space */

 147 extern ssize_t mdb_tgt_aread(mdb_tgt_t *, mdb_tgt_as_t,
 148         void *, size_t, mdb_tgt_addr_t);

 150 extern ssize_t mdb_tgt_awrite(mdb_tgt_t *, mdb_tgt_as_t,
 151         const void *, size_t, mdb_tgt_addr_t);

 153 extern ssize_t mdb_tgt_vread(mdb_tgt_t *, void *, size_t, uintptr_t);
 154 extern ssize_t mdb_tgt_vwrite(mdb_tgt_t *, const void *, size_t, uintptr_t);
 155 extern ssize_t mdb_tgt_pread(mdb_tgt_t *, void *, size_t, physaddr_t);
 156 extern ssize_t mdb_tgt_pwrite(mdb_tgt_t *, const void *, size_t, physaddr_t);
 157 extern ssize_t mdb_tgt_fread(mdb_tgt_t *, void *, size_t, uintptr_t);
 158 extern ssize_t mdb_tgt_fwrite(mdb_tgt_t *, const void *, size_t, uintptr_t);
 159 extern ssize_t mdb_tgt_ioread(mdb_tgt_t *, void *, size_t, uintptr_t);
 160 extern ssize_t mdb_tgt_iowrite(mdb_tgt_t *, const void *, size_t, uintptr_t);

 162 /*
 163  * Convert an address-space's virtual address to the corresponding
 164  * physical address (only useful for kernel targets):
 165  */
 166 extern int mdb_tgt_vtop(mdb_tgt_t *, mdb_tgt_as_t, uintptr_t, physaddr_t *);

 168 /*
 169  * Convenience functions for reading and writing null-terminated
 170  * strings from any of the target address spaces:
 171  */
 172 extern ssize_t mdb_tgt_readstr(mdb_tgt_t *, mdb_tgt_as_t,
 173         char *, size_t, mdb_tgt_addr_t);

 175 extern ssize_t mdb_tgt_writestr(mdb_tgt_t *, mdb_tgt_as_t,
 176         const char *, mdb_tgt_addr_t);

 178 /*
 179  * Symbol Table Interface
 180  *
 181  * Each target can provide access to one or more symbol tables, which can be
 182  * iterated over, or used to lookup symbols by either name or address.  The
```



```
 183  * target can support a primary executable and primary dynamic symbol table,
 184  * a symbol table for its run-time link-editor, and symbol tables for one or
 185  * more loaded objects.  A symbol is uniquely identified by an object name,
 186  * a symbol table id, and a symbol id.  Symbols can be discovered by iterating
 187  * over them, looking them up by name, or looking them up by address.
 188  */

 190 typedef struct mdb_syminfo {
 191         uint_t sym_table;       /* Symbol table id (see symbol_iter, below) */
 192         uint_t sym_id;          /* Symbol identifier */
 193 } mdb_syminfo_t;
_____unchanged_portion_omitted_

 334 /*
 335  * Program state (st_state):
 336  * (MDB_STATE_* definitions in the module API need to be in sync with these)
 337  */
 338 #define MDB_TGT_IDLE            0       /* Target is idle (not running yet) */
 339 #define MDB_TGT_RUNNING         1       /* Target is currently executing */
 340 #define MDB_TGT_STOPPED         2       /* Target is stopped */
 341 #define MDB_TGT_UNDEAD          3       /* Target is undead (zombie) */
 342 #define MDB_TGT_DEAD            4       /* Target is dead (core dump) */
 343 #define MDB_TGT_LOST            5       /* Target lost by debugger */

 345 /*
 346  * Status flags (st_flags):
 347  */
 348 #define MDB_TGT_ISTOP           0x1     /* Stop on event of interest */
 349 #define MDB_TGT_DSTOP           0x2     /* Stop directive is pending */
 350 #define MDB_TGT_BUSY            0x4     /* Busy in debugger */

 352 extern int mdb_tgt_status(mdb_tgt_t *, mdb_tgt_status_t *);
 353 extern int mdb_tgt_run(mdb_tgt_t *, int, const struct mdb_arg *);
 354 extern int mdb_tgt_step(mdb_tgt_t *, mdb_tgt_status_t *);
 355 extern int mdb_tgt_step_out(mdb_tgt_t *, mdb_tgt_status_t *);
 355 extern int mdb_tgt_step_branch(mdb_tgt_t *, mdb_tgt_status_t *);
 356 extern int mdb_tgt_next(mdb_tgt_t *, mdb_tgt_status_t *);
 357 extern int mdb_tgt_continue(mdb_tgt_t *, mdb_tgt_status_t *);
 358 extern int mdb_tgt_signal(mdb_tgt_t *, int);

 360 /*
 361  * Iterating through the specifier list yields the integer id (VID) and private
 362  * data pointer for each specifier.
 363  */
 364 typedef int mdb_tgt_vespec_f(mdb_tgt_t *, void *, int, void *);

 366 /*
 367  * Each event specifier is defined to be in one of the following states.  The
 368  * state transitions are discussed in detail in the comments in mdb_target.c.
 369  */
 370 #define MDB_TGT_SPEC_IDLE       1       /* Inactive (e.g. object not loaded) */
 371 #define MDB_TGT_SPEC_ACTIVE     2       /* Active but not armed in target */
 372 #define MDB_TGT_SPEC_ARMED      3       /* Active and armed (e.g. bkpt set) */
 373 #define MDB_TGT_SPEC_ERROR      4       /* Failed to arm event */

 375 /*
 376  * Event specifiers may also have one or more of the following additional
 377  * properties (spec_flags bits):
 378  */
 379 #define MDB_TGT_SPEC_INTERNAL   0x0001  /* Internal to target implementation */
 380 #define MDB_TGT_SPEC_SILENT     0x0002  /* Do not describe when matched */
 381 #define MDB_TGT_SPEC_TEMPORARY  0x0004  /* Delete next time target stops */
 382 #define MDB_TGT_SPEC_MATCHED    0x0008  /* Specifier matched at last stop */
 383 #define MDB_TGT_SPEC_DISABLED   0x0010  /* Specifier cannot be armed */
 384 #define MDB_TGT_SPEC_DELETED    0x0020  /* Specifier has been deleted */
 385 #define MDB_TGT_SPEC_AUTODEL    0x0040  /* Delete when match limit reached */
```

```
386 #define MDB_TGT_SPEC_AUTODIS    0x0080  /* Disable when match limit reached */
387 #define MDB_TGT_SPEC_AUTOSTOP   0x0100  /* Stop when match limit reached */
388 #define MDB_TGT_SPEC_STICKY     0x0200  /* Do not delete as part of :z */

390 #define MDB_TGT_SPEC_HIDDEN     (MDB_TGT_SPEC_INTERNAL | MDB_TGT_SPEC_SILENT)

392 typedef struct mdb_tgt_spec_desc {
393         int spec_id;                    /* Event specifier id (VID) */
394         uint_t spec_flags;              /* Flags (see above) */
395         uint_t spec_hits;               /* Count of number of times matched */
396         uint_t spec_limit;              /* Limit on number of times matched */
397         int spec_state;                 /* State (see above) */
398         int spec_errno;                 /* Last error code (if IDLE or ERROR) */
399         uintptr_t spec_base;            /* Start of affected memory region */
400         size_t spec_size;               /* Size of affected memory region */
401         void *spec_data;                /* Callback private data */
402 } mdb_tgt_spec_desc_t;
```

*unchanged_portion_omitted_*

```
**********************************************************
   14888 Fri Apr  6 17:24:55 2018
new/usr/src/cmd/mdb/common/mdb/mdb_target_impl.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2018, Joyent, Inc.  All rights reserved.
 * Copyright (c) 2012, Joyent, Inc.  All rights reserved.
 */

#ifndef _MDB_TARGET_IMPL_H
#define _MDB_TARGET_IMPL_H

#include <mdb/mdb_target.h>
#include <mdb/mdb_module.h>
#include <mdb/mdb_list.h>
#include <mdb/mdb_gelf.h>
#include <sys/auxv.h>

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _MDB

/*
 * Target Operations
 *
 * This ops vector implements the set of primitives which can be used by the
 * debugger to interact with the target, and encompasses most of the calls
 * found in <mdb/mdb_target.h>.  The remainder of the target interface is
 * implemented by common code that invokes these primitives or manipulates
 * the common target structures directly.
 */

typedef struct mdb_tgt_ops {
        int (*t_setflags)(mdb_tgt_t *, int);
        int (*t_setcontext)(mdb_tgt_t *, void *);
```



```
        void (*t_activate)(mdb_tgt_t *);
        void (*t_deactivate)(mdb_tgt_t *);
        void (*t_periodic)(mdb_tgt_t *);
        void (*t_destroy)(mdb_tgt_t *);

        const char *(*t_name)(mdb_tgt_t *);
        const char *(*t_isa)(mdb_tgt_t *);
        const char *(*t_platform)(mdb_tgt_t *);
        int (*t_uname)(mdb_tgt_t *, struct utsname *);
        int (*t_dmodel)(mdb_tgt_t *);

        ssize_t (*t_aread)(mdb_tgt_t *,
            mdb_tgt_as_t, void *, size_t, mdb_tgt_addr_t);

        ssize_t (*t_awrite)(mdb_tgt_t *,
            mdb_tgt_as_t, const void *, size_t, mdb_tgt_addr_t);

        ssize_t (*t_vread)(mdb_tgt_t *, void *, size_t, uintptr_t);
        ssize_t (*t_vwrite)(mdb_tgt_t *, const void *, size_t, uintptr_t);
        ssize_t (*t_pread)(mdb_tgt_t *, void *, size_t, physaddr_t);
        ssize_t (*t_pwrite)(mdb_tgt_t *, const void *, size_t, physaddr_t);
        ssize_t (*t_fread)(mdb_tgt_t *, void *, size_t, uintptr_t);
        ssize_t (*t_fwrite)(mdb_tgt_t *, const void *, size_t, uintptr_t);
        ssize_t (*t_ioread)(mdb_tgt_t *, void *, size_t, uintptr_t);
        ssize_t (*t_iowrite)(mdb_tgt_t *, const void *, size_t, uintptr_t);

        int (*t_vtop)(mdb_tgt_t *, mdb_tgt_as_t, uintptr_t, physaddr_t *);

        int (*t_lookup_by_name)(mdb_tgt_t *,
            const char *, const char *, GElf_Sym *, mdb_syminfo_t *);

        int (*t_lookup_by_addr)(mdb_tgt_t *,
            uintptr_t, uint_t, char *, size_t, GElf_Sym *, mdb_syminfo_t *);

        int (*t_symbol_iter)(mdb_tgt_t *,
            const char *, uint_t, uint_t, mdb_tgt_sym_f *, void *);

        int (*t_mapping_iter)(mdb_tgt_t *, mdb_tgt_map_f *, void *);
        int (*t_object_iter)(mdb_tgt_t *, mdb_tgt_map_f *, void *);

        const mdb_map_t *(*t_addr_to_map)(mdb_tgt_t *, uintptr_t);
        const mdb_map_t *(*t_name_to_map)(mdb_tgt_t *, const char *);
        struct ctf_file *(*t_addr_to_ctf)(mdb_tgt_t *, uintptr_t);
        struct ctf_file *(*t_name_to_ctf)(mdb_tgt_t *, const char *);

        int (*t_status)(mdb_tgt_t *, mdb_tgt_status_t *);
        int (*t_run)(mdb_tgt_t *, int, const struct mdb_arg *);
        int (*t_step)(mdb_tgt_t *, mdb_tgt_status_t *);
        int (*t_step_out)(mdb_tgt_t *, uintptr_t *);
        int (*t_step_branch)(mdb_tgt_t *);
        int (*t_next)(mdb_tgt_t *, uintptr_t *);
        int (*t_cont)(mdb_tgt_t *, mdb_tgt_status_t *);
        int (*t_signal)(mdb_tgt_t *, int);

        int (*t_add_vbrkpt)(mdb_tgt_t *, uintptr_t,
            int, mdb_tgt_se_f *, void *);
        int (*t_add_sbrkpt)(mdb_tgt_t *, const char *,
            int, mdb_tgt_se_f *, void *);

        int (*t_add_pwapt)(mdb_tgt_t *, physaddr_t, size_t, uint_t,
            int, mdb_tgt_se_f *, void *);
        int (*t_add_vwapt)(mdb_tgt_t *, uintptr_t, size_t, uint_t,
            int, mdb_tgt_se_f *, void *);
        int (*t_add_iowapt)(mdb_tgt_t *, uintptr_t, size_t, uint_t,
            int, mdb_tgt_se_f *, void *);
```

```
123         int (*t_add_sysenter)(mdb_tgt_t *, int, int, mdb_tgt_se_f *, void *);
124         int (*t_add_sysexit)(mdb_tgt_t *, int, int, mdb_tgt_se_f *, void *);
125         int (*t_add_signal)(mdb_tgt_t *, int, int, mdb_tgt_se_f *, void *);
126         int (*t_add_fault)(mdb_tgt_t *, int, int, mdb_tgt_se_f *, void *);

128         int (*t_getareg)(mdb_tgt_t *, mdb_tgt_tid_t, const char *,
129             mdb_tgt_reg_t *);
130         int (*t_putareg)(mdb_tgt_t *, mdb_tgt_tid_t, const char *,
131             mdb_tgt_reg_t);

133         int (*t_stack_iter)(mdb_tgt_t *, const mdb_tgt_gregset_t *,
134             mdb_tgt_stack_f *, void *);

136         int (*t_auxv)(mdb_tgt_t *, const auxv_t **auxvp);
137 } mdb_tgt_ops_t;
```

*unchanged_portion_omitted_*

```
**********************************************************
    6175 Fri Apr  6 17:24:55 2018
new/usr/src/cmd/mdb/common/mdb/mdb_value.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

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

/*
 * Immediate Value Target
 *
 * The immediate value target is used when the '=' verb is used to
 * format an immediate value, or with ::print -i.  The target is
 * initialized with a specific value, and then simply copies bytes from
 * this integer in its read routine.  Two notes:
 *
 * (1) the address parameter of value_read is treated as an offset into
 * the immediate value.
 *
 * (2) on big-endian systems, we need to be careful about the place we
 * copy data from. If the caller specified a typesize in the argv array
 * we use that for offsetting, otherwise we use the read size.
 * If the user didn't specify the typesize, then 'addr' is ignored,
 * and all reads are at an offset of 0 into the immediate value. This
 * covers both the usage of ::print -i, and the semantics of adb
 * commands like "0x1234=X", which should produce 0x1234 as a result;
 * the adb model is for it to act like a cast down to the smaller
 * integer type; this is handled as mentioned.
 */

#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_types.h>
#include <mdb/mdb_conf.h>
#include <mdb/mdb_err.h>

#include <sys/isa_defs.h>
```



```
#include <strings.h>

void mdb_value_tgt_destroy(mdb_tgt_t *);

typedef struct mdb_value_data {
        uintmax_t mvd_data;
        size_t mvd_typesize;
} mdb_value_data_t;
______unchanged_portion_omitted_

static const mdb_tgt_ops_t value_ops = {
        (int (*)()) mdb_tgt_notsup,             /* t_setflags */
        (int (*)()) mdb_tgt_notsup,             /* t_setcontext */
        (void (*)()) mdb_tgt_nop,               /* t_activate */
        (void (*)()) mdb_tgt_nop,               /* t_deactivate */
        (void (*)()) mdb_tgt_nop,               /* t_periodic */
        mdb_value_tgt_destroy,                  /* t_destroy */
        (const char *(*)()) mdb_tgt_null,       /* t_name */
        (const char *(*)()) mdb_conf_isa,       /* t_isa */
        (const char *(*)()) mdb_conf_platform,  /* t_platform */
        (int (*)()) mdb_tgt_notsup,             /* t_uname */
        (int (*)()) mdb_tgt_notsup,             /* t_dmodel */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_aread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_awrite */
        value_read,                             /* t_vread */
        value_write,                            /* t_vwrite */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_pread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_pwrite */
        value_read,                             /* t_fread */
        value_write,                            /* t_fwrite */
        value_read,                             /* t_ioread */
        value_write,                            /* t_iowrite */
        (int (*)()) mdb_tgt_notsup,             /* t_vtop */
        (int (*)()) mdb_tgt_notsup,             /* t_lookup_by_name */
        (int (*)()) mdb_tgt_notsup,             /* t_lookup_by_addr */
        (int (*)()) mdb_tgt_notsup,             /* t_symbol_iter */
        (int (*)()) mdb_tgt_notsup,             /* t_mapping_iter */
        (int (*)()) mdb_tgt_notsup,             /* t_object_iter */
        (const mdb_map_t *(*)()) mdb_tgt_null,  /* t_addr_to_map */
        (const mdb_map_t *(*)()) mdb_tgt_null,  /* t_name_to_map */
        (struct ctf_file *(*)()) mdb_tgt_null,  /* t_addr_to_ctf */
        (struct ctf_file *(*)()) mdb_tgt_null,  /* t_name_to_ctf */
        (int (*)()) mdb_tgt_notsup,             /* t_status */
        (int (*)()) mdb_tgt_notsup,             /* t_run */
        (int (*)()) mdb_tgt_notsup,             /* t_step */
        (int (*)()) mdb_tgt_notsup,             /* t_step_out */
        (int (*)()) mdb_tgt_notsup,             /* t_step_branch */
        (int (*)()) mdb_tgt_notsup,             /* t_next */
        (int (*)()) mdb_tgt_notsup,             /* t_cont */
        (int (*)()) mdb_tgt_notsup,             /* t_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_vbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_sbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_pwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_vwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_iowapt */
        (int (*)()) mdb_tgt_null,               /* t_add_sysenter */
        (int (*)()) mdb_tgt_null,               /* t_add_sysexit */
        (int (*)()) mdb_tgt_null,               /* t_add_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_fault */
        (int (*)()) mdb_tgt_notsup,             /* t_getareg */
        (int (*)()) mdb_tgt_notsup,             /* t_putareg */
        (int (*)()) mdb_tgt_nop,                /* t_stack_iter */
        (int (*)()) mdb_tgt_notsup              /* t_auxv */
};
______unchanged_portion_omitted_
```

```
**********************************************************
   23287 Fri Apr  6 17:24:55 2018
new/usr/src/cmd/mdb/i86pc/modules/unix/i86mmu.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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.
 */

/*
 * This part of the file contains the mdb support for dcmds:
 *      ::memseg_list
 * and walkers for:
 *      memseg - a memseg list walker for ::memseg_list
 *
 */

#include <sys/types.h>
#include <sys/machparam.h>
#include <sys/controlregs.h>
#include <sys/mach_mmu.h>
#ifdef __xpv
#include <sys/hypervisor.h>
#endif
#include <vm/as.h>

#include <mdb/mdb_modapi.h>
#include <mdb/mdb_target.h>

#include <vm/page.h>
#include <vm/hat_i86.h>

#define VA_SIGN_BIT (1UL << 47)
#define VA_SIGN_EXTEND(va) (((va) ^ VA_SIGN_BIT) - VA_SIGN_BIT)

struct pfn2pp {
        pfn_t pfn;
        page_t *pp;
};
```

*unchanged_portion_omitted_*


```
/*
 * Print a PTE in more human friendly way. The PTE is assumed to be in
 * a level 0 page table, unless -l specifies another level.
 *
 * The PTE value can be specified as the -p option, since on a 32 bit kernel
 * with PAE running it's larger than a uintptr_t.
 */
static int
do_pte_dcmd(int level, uint64_t pte)
{
        static char *attr[] = {
            "wrback", "wrthru", "uncached", "uncached",
            "wrback", "wrthru", "wrcombine", "uncached"};
        int pat_index = 0;
        pfn_t mfn;

        mdb_printf("pte=0x%llr: ", pte);
        mdb_printf("pte=%llr: ", pte);
        if (PTE_GET(pte, mmu.pt_nx))
                mdb_printf("noexec ");

        mfn = pte2mfn(pte, level);
        mdb_printf("%s=0x%lr ", is_xpv ? "mfn" : "pfn", mfn);

        if (PTE_GET(pte, mmu.pt_nx))
                mdb_printf("noexec ");

        if (PTE_GET(pte, PT_NOCONSIST))
                mdb_printf("noconsist ");

        if (PTE_GET(pte, PT_NOSYNC))
                mdb_printf("nosync ");

        if (PTE_GET(pte, mmu.pt_global))
                mdb_printf("global ");

        if (level > 0 && PTE_GET(pte, PT_PAGESIZE))
                mdb_printf("largepage ");

        if (level > 0 && PTE_GET(pte, PT_MOD))
                mdb_printf("mod ");

        if (level > 0 && PTE_GET(pte, PT_REF))
                mdb_printf("ref ");

        if (PTE_GET(pte, PT_USER))
                mdb_printf("user ");

        if (PTE_GET(pte, PT_WRITABLE))
                mdb_printf("write ");

        /*
         * Report non-standard cacheability
         */
        pat_index = 0;
        if (level > 0) {
                if (PTE_GET(pte, PT_PAGESIZE) && PTE_GET(pte, PT_PAT_LARGE))
                        pat_index += 4;
        } else {
                if (PTE_GET(pte, PT_PAT_4K))
                        pat_index += 4;
        }

        if (PTE_GET(pte, PT_NOCACHE))
                pat_index += 2;
```

```
462         if (PTE_GET(pte, PT_WRITETHRU))
463                 pat_index += 1;
464
465         if (pat_index != 0)
466                 mdb_printf("%s", attr[pat_index]);
467
468         if (PTE_GET(pte, PT_VALID) == 0)
469                 mdb_printf(" !VALID ");
470
471         mdb_printf("\n");
472         return (DCMD_OK);
473 }
474
475 /*
476  * Print a PTE in more human friendly way. The PTE is assumed to be in
477  * a level 0 page table, unless -l specifies another level.
478  *
479  * The PTE value can be specified as the -p option, since on a 32 bit kernel
480  * with PAE running it's larger than a uintptr_t.
481  */
482 /*ARGSUSED*/
483 int
484 pte_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
485 {
486         uint64_t level = 0;
487         int level = 0;
488         uint64_t pte = 0;
489         char *level_str = NULL;
490         char *pte_str = NULL;
491
492         init_mmu();
493
494         if (mmu.num_level == 0)
495                 return (DCMD_ERR);
496
497         if ((flags & DCMD_ADDRSPEC) == 0)
498         if (mdb_getopts(argc, argv,
499             'p', MDB_OPT_STR, &pte_str,
500             'l', MDB_OPT_STR, &level_str) != argc)
501                 return (DCMD_USAGE);
502
503         if (mdb_getopts(argc, argv,
504             'l', MDB_OPT_UINT64, &level) != argc)
505         /*
506          * parse the PTE to decode, if it's 0, we don't do anything
507          */
508         if (pte_str != NULL) {
509                 pte = mdb_strtoull(pte_str);
510         } else {
511                 if ((flags & DCMD_ADDRSPEC) == 0)
512                 return (DCMD_USAGE);
513                 pte = addr;
514         }
515         if (pte == 0)
516                 return (DCMD_OK);
517
518         if (level > mmu.max_level) {
519                 mdb_warn("invalid level %lu\n", level);
520         /*
521          * parse the level if supplied
522          */
523         if (level_str != NULL) {
524                 level = mdb_strtoull(level_str);
525                 if (level < 0 || level > mmu.max_level)
526                 return (DCMD_ERR);
527         }
```



```
528         if (addr == 0)
529                 return (DCMD_OK);
530
531         return (do_pte_dcmd((int)level, addr));
532         return (do_pte_dcmd(level, pte));
533 }
_____unchanged_portion_omitted_

534 static x86pte_t
535 get_pte(hat_t *hat, htable_t *htable, uintptr_t addr)
536 {
537         x86pte_t buf;
538         x86pte32_t *pte32 = (x86pte32_t *)&buf;
539         size_t len;
540
541         if (htable->ht_flags & HTABLE_COPIED) {
542                 uintptr_t ptr = (uintptr_t)hat->hat_copied_ptes;
543         if (htable->ht_flags & HTABLE_VLP) {
544                 uintptr_t ptr = (uintptr_t)hat->hat_vlp_ptes;
545                 ptr += va2entry(htable, addr) << mmu.pte_size_shift;
546                 return (*(x86pte_t *)ptr);
547         }
548
549                 len = mdb_vread(&buf, mmu.pte_size, ptr);
550         } else {
551         paddr_t paddr = mmu_ptob((paddr_t)htable->ht_pfn);
552         paddr += va2entry(htable, addr) << mmu.pte_size_shift;
553                 len = mdb_pread(&buf, mmu.pte_size, paddr);
554         }
555
556         if ((mdb_pread(&buf, mmu.pte_size, paddr)) == mmu.pte_size)
557                 return (buf);
558
559         if (len != mmu.pte_size)
560         return (0);
561
562         if (mmu.pte_size == sizeof (x86pte_t))
563                 return (buf);
564         return (*pte32);
565 }
566
567 static int
568 do_va2pa(uintptr_t addr, struct as *asp, int print_level, physaddr_t *pap,
569     pfn_t *mfnp)
570 {
571         struct as as;
572         struct hat *hatp;
573         struct hat hat;
574         htable_t *ht;
575         htable_t htable;
576         uintptr_t base;
577         int h;
578         int level;
579         int found = 0;
580         x86pte_t pte;
581         physaddr_t paddr;
582
583         if (asp != NULL) {
584                 if (mdb_vread(&as, sizeof (as), (uintptr_t)asp) == -1) {
585                         mdb_warn("Couldn't read struct as\n");
586                         return (DCMD_ERR);
587                 }
588                 hatp = as.a_hat;
589         } else {
590                 hatp = khat;
```

```
561         }

563         /*
564          * read the hat and its hash table
565          */
566         if (mdb_vread(&hat, sizeof (hat), (uintptr_t)hatp) == -1) {
567                 mdb_warn("Couldn't read struct hat\n");
568                 return (DCMD_ERR);
569         }

571         /*
572          * read the htable hashtable
573          */
574         for (level = 0; level <= mmu.max_level; ++level) {
575                 if (level == TOP_LEVEL(&hat))
576                         base = 0;
577                 else
578                         base = addr & mmu.level_mask[level + 1];

580                 for (h = 0; h < hat.hat_num_hash; ++h) {
581                         if (mdb_vread(&ht, sizeof (htable_t *),
582                             (uintptr_t)(hat.hat_ht_hash + h)) == -1) {
583                                 mdb_warn("Couldn't read htable\n");
584                                 return (DCMD_ERR);
585                         }
586                         for (; ht != NULL; ht = htable.ht_next) {
587                                 if (mdb_vread(&htable, sizeof (htable_t),
588                                     (uintptr_t)ht) == -1) {
589                                         mdb_warn("Couldn't read htable\n");
590                                         return (DCMD_ERR);
591                                 }

593                                 if (htable.ht_vaddr != base ||
594                                     htable.ht_level != level)
595                                         continue;

597                                 pte = get_pte(&hat, &htable, addr);

599                                 if (print_level) {
600                                         mdb_printf("\tlevel=%d htable=0x%p "
601                                             "pte=0x%llr\n", level, ht, pte);
624                                         mdb_printf("\tlevel=%d htable=%p "
625                                             "pte=%llr\n", level, ht, pte);
602                                 }

604                                 if (!PTE_ISVALID(pte)) {
605                                         mdb_printf("Address %p is unmapped.\n",
606                                             addr);
607                                         return (DCMD_ERR);
608                                 }

610                                 if (found)
611                                         continue;

613                                 if (PTE_IS_LGPG(pte, level))
614                                         paddr = mdb_ma_to_pa(pte &
615                                             PT_PADDR_LGPG);
616                                 else
617                                         paddr = mdb_ma_to_pa(pte & PT_PADDR);
618                                 paddr += addr & mmu.level_offset[level];
619                                 if (pap != NULL)
620                                         *pap = paddr;
621                                 if (mfnp != NULL)
622                                         *mfnp = pte2mfn(pte, level);
623                                 found = 1;
624                         }
```



```
625                 }
626         }

628 done:
629         if (!found)
630                 return (DCMD_ERR);
631         return (DCMD_OK);
632 }
```

*unchanged_portion_omitted_*

```
689 /*
690  * Report all hat's that either use PFN as a page table or that map the page.
691  */
692 static int
693 do_report_maps(pfn_t pfn)
694 {
695         struct hat *hatp;
696         struct hat hat;
697         htable_t *ht;
698         htable_t htable;
699         uintptr_t base;
700         int h;
701         int level;
702         int entry;
703         x86pte_t pte;
728         x86pte_t buf;
729         x86pte32_t *pte32 = (x86pte32_t *)&buf;
704         physaddr_t paddr;
705         size_t len;

707         /*
708          * The hats are kept in a list with khat at the head.
709          */
710         for (hatp = khat; hatp != NULL; hatp = hat.hat_next) {
711                 /*
712                  * read the hat and its hash table
713                  */
714                 if (mdb_vread(&hat, sizeof (hat), (uintptr_t)hatp) == -1) {
715                         mdb_warn("Couldn't read struct hat\n");
716                         return (DCMD_ERR);
717                 }

719                 /*
720                  * read the htable hashtable
721                  */
722                 paddr = 0;
723                 for (h = 0; h < hat.hat_num_hash; ++h) {
724                         if (mdb_vread(&ht, sizeof (htable_t *),
725                             (uintptr_t)(hat.hat_ht_hash + h)) == -1) {
726                                 mdb_warn("Couldn't read htable\n");
727                                 return (DCMD_ERR);
728                         }
729                         for (; ht != NULL; ht = htable.ht_next) {
730                                 if (mdb_vread(&htable, sizeof (htable_t),
731                                     (uintptr_t)ht) == -1) {
732                                         mdb_warn("Couldn't read htable\n");
733                                         return (DCMD_ERR);
734                                 }

736                                 /*
737                                  * only report kernel addresses once
738                                  */
739                                 if (hatp != khat &&
740                                     htable.ht_vaddr >= kernelbase)
741                                         continue;
```

```
                                /*
                                 * Is the PFN a pagetable itself?
                                 */
                                if (htable.ht_pfn == pfn) {
                                        mdb_printf("Pagetable for "
                                            "hat=%p htable=%p\n", hatp, ht);
                                        continue;
                                }

                                /*
                                 * otherwise, examine page mappings
                                 */
                                level = htable.ht_level;
                                if (level > mmu.max_page_level)
                                        continue;
                                paddr = mmu_ptob((physaddr_t)htable.ht_pfn);
                                for (entry = 0;
                                    entry < HTABLE_NUM_PTES(&htable);
                                    ++entry) {

                                        base = htable.ht_vaddr + entry *
                                            mmu.level_size[level];

                                        /*
                                         * only report kernel addresses once
                                         */
                                        if (hatp != khat &&
                                            base >= kernelbase)
                                                continue;

                                        len = mdb_pread(&pte, mmu.pte_size,
                                        len = mdb_pread(&buf, mmu.pte_size,
                                            paddr + entry * mmu.pte_size);
                                        if (len != mmu.pte_size)
                                                return (DCMD_ERR);
                                        if (mmu.pte_size == sizeof (x86pte_t))
                                                pte = buf;
                                        else
                                                pte = *pte32;

                                        if ((pte & PT_VALID) == 0)
                                                continue;
                                        if (level == 0 || !(pte & PT_PAGESIZE))
                                                pte &= PT_PADDR;
                                        else
                                                pte &= PT_PADDR_LGPG;
                                        if (mmu_btop(mdb_ma_to_pa(pte)) != pfn)
                                                continue;
                                        mdb_printf("hat=%p maps addr=%p\n",
                                            hatp, (caddr_t)base);
                                }
                        }
                }
        }

done:
        return (DCMD_OK);
}
_____unchanged_portion_omitted_

static int
do_ptable_dcmd(pfn_t pfn, uint64_t level)
do_ptable_dcmd(pfn_t pfn)
{
        struct hat *hatp;
        struct hat hat;
```



```
        htable_t *ht;
        htable_t htable;
        uintptr_t base;
        int h;
        int level;
        int entry;
        uintptr_t pagesize;
        x86pte_t pte;
        x86pte_t buf;
        x86pte32_t *pte32 = (x86pte32_t *)&buf;
        physaddr_t paddr;
        size_t len;

        /*
         * The hats are kept in a list with khat at the head.
         */
        for (hatp = khat; hatp != NULL; hatp = hat.hat_next) {
                /*
                 * read the hat and its hash table
                 */
                if (mdb_vread(&hat, sizeof (hat), (uintptr_t)hatp) == -1) {
                        mdb_warn("Couldn't read struct hat\n");
                        return (DCMD_ERR);
                }

                /*
                 * read the htable hashtable
                 */
                paddr = 0;
                for (h = 0; h < hat.hat_num_hash; ++h) {
                        if (mdb_vread(&ht, sizeof (htable_t *),
                            (uintptr_t)(hat.hat_ht_hash + h)) == -1) {
                                mdb_warn("Couldn't read htable\n");
                                return (DCMD_ERR);
                        }
                        for (; ht != NULL; ht = htable.ht_next) {
                                if (mdb_vread(&htable, sizeof (htable_t),
                                    (uintptr_t)ht) == -1) {
                                        mdb_warn("Couldn't read htable\n");
                                        return (DCMD_ERR);
                                }

                                /*
                                 * Is this the PFN for this htable
                                 */
                                if (htable.ht_pfn == pfn)
                                        goto found_it;
                        }
                }
        }

found_it:
        if (htable.ht_pfn == pfn) {
                mdb_printf("htable=%p\n", ht);
                if (level == (uint64_t)-1) {
                        level = htable.ht_level;
                } else if (htable.ht_level != level) {
                        mdb_warn("htable has level %d but forcing level %lu\n",
                            htable.ht_level, level);
                }
                base = htable.ht_vaddr;
                pagesize = mmu.level_size[level];
        } else {
                if (level == (uint64_t)-1)
                        level = 0;
                mdb_warn("couldn't find matching htable, using level=%lu, "
```

```
895                     "base address=0x0\n", level);
919                 mdb_printf("Unknown pagetable - assuming level/addr 0");
920                 level = 0;      /* assume level == 0 for PFN */
896                 base = 0;
897                 pagesize = mmu.level_size[level];
922                 pagesize = MMU_PAGESIZE;
898         }

900         paddr = mmu_ptob((physaddr_t)pfn);
901         for (entry = 0; entry < mmu.ptes_per_table; ++entry) {
902                 len = mdb_pread(&buf, mmu.pte_size,
903                     paddr + entry * mmu.pte_size);
904                 if (len != mmu.pte_size)
905                         return (DCMD_ERR);
931                 if (mmu.pte_size == sizeof (x86pte_t))
906                         pte = buf;
933                 else
934                         pte = *pte32;

908                 if (pte == 0)
909                         continue;

911                 mdb_printf("[%3d] va=0x%p ", entry,
912                     VA_SIGN_EXTEND(base + entry * pagesize));
939                 mdb_printf("[%3d] va=%p ", entry, base + entry * pagesize);
913                 do_pte_dcmd(level, pte);
914         }

916 done:
917         return (DCMD_OK);
918 }

920 /*
921  * Dump the page table at the given PFN
922  */
923 /*ARGSUSED*/
924 int
925 ptable_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
926 {
927         pfn_t pfn;
928         uint_t mflag = 0;
929         uint64_t level = (uint64_t)-1;

931         init_mmu();

933         if (mmu.num_level == 0)
934                 return (DCMD_ERR);

936         if ((flags & DCMD_ADDRSPEC) == 0)
937                 return (DCMD_USAGE);

939         if (mdb_getopts(argc, argv,
940             'm', MDB_OPT_SETBITS, TRUE, &mflag,
941             'l', MDB_OPT_UINT64, &level, NULL) != argc)
966             'm', MDB_OPT_SETBITS, TRUE, &mflag, NULL) != argc)
942                 return (DCMD_USAGE);

944         if (level != (uint64_t)-1 && level > mmu.max_level) {
945                 mdb_warn("invalid level %lu\n", level);
946                 return (DCMD_ERR);
947         }

949         pfn = (pfn_t)addr;
950         if (mflag)
951                 pfn = mdb_mfn_to_pfn(pfn);
```



```
953         return (do_ptable_dcmd(pfn, level));
973         return (do_ptable_dcmd(pfn));
954 }
_____unchanged_portion_omitted_

993 /*
994  * Dump the htables for the given hat
995  */
996 /*ARGSUSED*/
997 int
998 htables_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
999 {
1000         hat_t *hat;

1002         init_mmu();

1004         if (mmu.num_level == 0)
1005                 return (DCMD_ERR);

1007         if ((flags & DCMD_ADDRSPEC) == 0)
1008                 return (DCMD_USAGE);

1010         hat = (hat_t *)addr;

1012         return (do_htables_dcmd(hat));
1013 }

1015 static uintptr_t
1016 entry2va(size_t *entries)
1017 {
1018         uintptr_t va = 0;

1020         for (level_t l = mmu.max_level; l >= 0; l--)
1021                 va += entries[l] << mmu.level_shift[l];

1023         return (VA_SIGN_EXTEND(va));
1024 }

1026 static void
1027 ptmap_report(size_t *entries, uintptr_t start,
1028     boolean_t user, boolean_t writable, boolean_t wflag)
1029 {
1030         uint64_t curva = entry2va(entries);

1032         mdb_printf("mapped %s,%s range of %lu bytes: %a-%a\n",
1033             user ? "user" : "kernel", writable ? "writable" : "read-only",
1034             curva - start, start, curva - 1);
1035         if (wflag && start >= kernelbase)
1036                 (void) mdb_call_dcmd("whatis", start, DCMD_ADDRSPEC, 0, NULL);
1037 }

1039 int
1040 ptmap_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
1041 {
1042         physaddr_t paddrs[MAX_NUM_LEVEL] = { 0, };
1043         size_t entry[MAX_NUM_LEVEL] = { 0, };
1044         uintptr_t start = (uintptr_t)-1;
1045         boolean_t writable = B_FALSE;
1046         boolean_t user = B_FALSE;
1047         boolean_t wflag = B_FALSE;
1048         level_t curlevel;

1050         if ((flags & DCMD_ADDRSPEC) == 0)
1051                 return (DCMD_USAGE);

1053         if (mdb_getopts(argc, argv,
```

```
1054             'w', MDB_OPT_SETBITS, TRUE, &wflag, NULL) != argc)
1055                 return (DCMD_USAGE);

1057         init_mmu();

1059         if (mmu.num_level == 0)
1060                 return (DCMD_ERR);

1062         curlevel = mmu.max_level;

1064         paddrs[curlevel] = addr & MMU_PAGEMASK;

1066         for (;;) {
1067                 physaddr_t pte_addr;
1068                 x86pte_t pte;

1070                 pte_addr = paddrs[curlevel] +
1071                     (entry[curlevel] << mmu.pte_size_shift);

1073                 if (mdb_pread(&pte, sizeof (pte), pte_addr) != sizeof (pte)) {
1074                         mdb_warn("couldn't read pte at %p", pte_addr);
1075                         return (DCMD_ERR);
1076                 }

1078                 if (PTE_GET(pte, PT_VALID) == 0) {
1079                         if (start != (uintptr_t)-1) {
1080                                 ptmap_report(entry, start,
1081                                     user, writable, wflag);
1082                                 start = (uintptr_t)-1;
1083                         }
1084                 } else if (curlevel == 0 || PTE_GET(pte, PT_PAGESIZE)) {
1085                         if (start == (uintptr_t)-1) {
1086                                 start = entry2va(entry);
1087                                 user = PTE_GET(pte, PT_USER);
1088                                 writable = PTE_GET(pte, PT_WRITABLE);
1089                         } else if (user != PTE_GET(pte, PT_USER) ||
1090                             writable != PTE_GET(pte, PT_WRITABLE)) {
1091                                 ptmap_report(entry, start,
1092                                     user, writable, wflag);
1093                                 start = entry2va(entry);
1094                                 user = PTE_GET(pte, PT_USER);
1095                                 writable = PTE_GET(pte, PT_WRITABLE);
1096                         }
1097                 } else {
1098                         /* Descend a level. */
1099                         physaddr_t pa = mmu_ptob(pte2mfn(pte, curlevel));
1100                         paddrs[--curlevel] = pa;
1101                         entry[curlevel] = 0;
1102                         continue;
1103                 }

1105                 while (++entry[curlevel] == mmu.ptes_per_table) {
1106                         /* Ascend back up. */
1107                         entry[curlevel] = 0;
1108                         if (curlevel == mmu.max_level) {
1109                                 if (start != (uintptr_t)-1) {
1110                                         ptmap_report(entry, start,
1111                                             user, writable, wflag);
1112                                 }
1113                                 goto out;
1114                         }

1116                         curlevel++;
1117                 }
1118         }
```



```
1120 out:
1121         return (DCMD_OK);
1122 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    2075 Fri Apr  6 17:24:55 2018
new/usr/src/cmd/mdb/i86pc/modules/unix/i86mmu.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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.
 */

#ifndef _I86MMU_H
#define _I86MMU_H

#ifdef  __cplusplus
extern "C" {
#endif

extern int pte_dcmd(uintptr_t addr, uint_t flags, int argc,
        const mdb_arg_t *argv);

extern int report_maps_dcmd(uintptr_t addr, uint_t flags, int argc,
        const mdb_arg_t *argv);

extern int htables_dcmd(uintptr_t addr, uint_t flags, int argc,
        const mdb_arg_t *argv);

extern int ptable_dcmd(uintptr_t addr, uint_t flags, int argc,
        const mdb_arg_t *argv);

extern int ptmap_dcmd(uintptr_t addr, uint_t flags, int argc,
        const mdb_arg_t *argv);

extern int va2pfn_dcmd(uintptr_t addr, uint_t flags, int argc,
        const mdb_arg_t *argv);

extern int mfntopfn_dcmd(uintptr_t addr, uint_t flags, int argc,
        const mdb_arg_t *argv);

extern int pfntomfn_dcmd(uintptr_t addr, uint_t flags, int argc,
        const mdb_arg_t *argv);

extern int memseg_list(uintptr_t addr, uint_t flags, int argc,
```



```
        const mdb_arg_t *argv);

extern int memseg_walk_init(mdb_walk_state_t *);
extern int memseg_walk_step(mdb_walk_state_t *);
extern void memseg_walk_fini(mdb_walk_state_t *);

extern void free_mmu(void);

#ifdef  __cplusplus
}
_____unchanged_portion_omitted_
```

```
**********************************************************
   25628 Fri Apr  6 17:24:56 2018
new/usr/src/cmd/mdb/i86pc/modules/unix/unix.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2018 Joyent, Inc.
 * Copyright 2015 Joyent, Inc.
 */

#include <mdb/mdb_modapi.h>
#include <mdb/mdb_ctf.h>
#include <sys/cpuvar.h>
#include <sys/systm.h>
#include <sys/traptrace.h>
#include <sys/x_call.h>
#include <sys/xc_levels.h>
#include <sys/avintr.h>
#include <sys/systm.h>
#include <sys/trap.h>
#include <sys/mutex.h>
#include <sys/mutex_impl.h>
#include "i86mmu.h"
#include "unix_sup.h"
#include <sys/apix.h>
#include <sys/x86_archext.h>
#include <sys/bitmap.h>
#include <sys/controlregs.h>

#define TT_HDLR_WIDTH   17


/* apix only */
static apix_impl_t *d_apixs[NCPU];
static int use_apix = 0;

static int
ttrace_ttr_size_check(void)
```



```
{
        mdb_ctf_id_t ttrtid;
        ssize_t ttr_size;

        if (mdb_ctf_lookup_by_name("trap_trace_rec_t", &ttrtid) != 0 ||
            mdb_ctf_type_resolve(ttrtid, &ttrtid) != 0) {
                mdb_warn("failed to determine size of trap_trace_rec_t; "
                    "non-TRAPTRACE kernel?\n");
                return (0);
        }

        if ((ttr_size = mdb_ctf_type_size(ttrtid)) !=
            sizeof (trap_trace_rec_t)) {
                /*
                 * On Intel machines, this will happen when TTR_STACK_DEPTH
                 * is changed.  This code could be smarter, and could
                 * dynamically adapt to different depths, but not until a
                 * need for such adaptation is demonstrated.
                 */
                mdb_warn("size of trap_trace_rec_t (%d bytes) doesn't "
                    "match expected %d\n", ttr_size, sizeof (trap_trace_rec_t));
                return (0);
        }

        return (1);
}
______unchanged_portion_omitted_

typedef struct ttrace_dcmd {
        processorid_t ttd_cpu;
        uint_t ttd_extended;
        uintptr_t ttd_kthread;
        trap_trace_ctl_t ttd_ttc[NCPU];
} ttrace_dcmd_t;

#if defined(__amd64)

#define DUMP(reg) #reg, regs->r_##reg
#define THREEREGS       "         %3s: %16lx %3s: %16lx %3s: %16lx\n"

static void
ttrace_dumpregs(trap_trace_rec_t *rec)
{
        struct regs *regs = &rec->ttr_regs;

        mdb_printf(THREEREGS, DUMP(rdi), DUMP(rsi), DUMP(rdx));
        mdb_printf(THREEREGS, DUMP(rcx), DUMP(r8), DUMP(r9));
        mdb_printf(THREEREGS, DUMP(rax), DUMP(rbx), DUMP(rbp));
        mdb_printf(THREEREGS, DUMP(r10), DUMP(r11), DUMP(r12));
        mdb_printf(THREEREGS, DUMP(r13), DUMP(r14), DUMP(r15));
        mdb_printf(THREEREGS, DUMP(ds), DUMP(es), DUMP(fs));
        mdb_printf(THREEREGS, DUMP(gs), "trp", regs->r_trapno, DUMP(err));
        mdb_printf(THREEREGS, DUMP(rip), DUMP(cs), DUMP(rfl));
        mdb_printf(THREEREGS, DUMP(rsp), DUMP(ss), "cr2", rec->ttr_cr2);
        mdb_printf("         %3s: %16lx %3s: %16lx\n",
            "fsb", regs->__r_fsbase,
            "gsb", regs->__r_gsbase);
        mdb_printf("\n");
}
______unchanged_portion_omitted_

#endif  /* __amd64 */

int
ttrace_walk(uintptr_t addr, trap_trace_rec_t *rec, ttrace_dcmd_t *dcmd)
{
```

```
466         struct regs *regs = &rec->ttr_regs;
467         processorid_t cpu = -1, i;
468
469         for (i = 0; i < NCPU; i++) {
470                 if (addr >= dcmd->ttd_ttc[i].ttc_first &&
471                     addr < dcmd->ttd_ttc[i].ttc_limit) {
472                         cpu = i;
473                         break;
474                 }
475         }
476
477         if (cpu == -1) {
478                 mdb_warn("couldn't find %p in any trap trace ctl\n", addr);
479                 return (WALK_ERR);
480         }
481
482         if (dcmd->ttd_cpu != -1 && cpu != dcmd->ttd_cpu)
483                 return (WALK_NEXT);
484
485         if (dcmd->ttd_kthread != 0 &&
486             dcmd->ttd_kthread != rec->ttr_curthread)
487                 return (WALK_NEXT);
488
489         mdb_printf("%3d %15llx ", cpu, rec->ttr_stamp);
490
491         for (i = 0; ttrace_hdlr[i].t_hdlr != NULL; i++) {
492                 if (rec->ttr_marker != ttrace_hdlr[i].t_marker)
493                         continue;
494                 mdb_printf("%4s ", ttrace_hdlr[i].t_name);
495                 if (ttrace_hdlr[i].t_hdlr(rec) == -1)
496                         return (WALK_ERR);
497         }
498
499         mdb_printf(" %a\n", regs->r_pc);
500
501         if (dcmd->ttd_extended == FALSE)
502                 return (WALK_NEXT);
503
504         if (rec->ttr_marker == TT_INTERRUPT)
505                 ttrace_intr_detail(rec);
506         else
507                 ttrace_dumpregs(rec);
508
509         if (rec->ttr_sdepth > 0) {
510                 for (i = 0; i < rec->ttr_sdepth; i++) {
511                         if (i >= TTR_STACK_DEPTH) {
512                                 mdb_printf("%17s*** invalid ttr_sdepth (is %d, "
513                                     "should be <= %d)\n", " ", rec->ttr_sdepth,
514                                     TTR_STACK_DEPTH);
515                                 break;
516                         }
517
518                         mdb_printf("%17s %a()\n", " ", rec->ttr_stack[i]);
519                 }
520                 mdb_printf("\n");
521         }
522
523         return (WALK_NEXT);
524 }
525
526 int
527 ttrace(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
528 {
529         ttrace_dcmd_t dcmd;
530         trap_trace_ctl_t *ttc = dcmd.ttd_ttc;
531         trap_trace_rec_t rec;
```



```
532         size_t ttc_size = sizeof (trap_trace_ctl_t) * NCPU;
533
534         if (!ttrace_ttr_size_check())
535                 return (WALK_ERR);
536
537         bzero(&dcmd, sizeof (dcmd));
538         dcmd.ttd_cpu = -1;
539         dcmd.ttd_extended = FALSE;
540
541         if (mdb_readsym(ttc, ttc_size, "trap_trace_ctl") == -1) {
542                 mdb_warn("symbol 'trap_trace_ctl' not found; "
543                     "non-TRAPTRACE kernel?\n");
544                 return (DCMD_ERR);
545         }
546
547         if (mdb_getopts(argc, argv,
548             'x', MDB_OPT_SETBITS, TRUE, &dcmd.ttd_extended,
549             't', MDB_OPT_UINTPTR, &dcmd.ttd_kthread, NULL) != argc)
550                 return (DCMD_USAGE);
551
552         if (DCMD_HDRSPEC(flags)) {
553                 mdb_printf("%3s %15s %4s %2s %-*s%s\n", "CPU",
554                     "TIMESTAMP", "TYPE", "Vec", TT_HDLR_WIDTH, "HANDLER",
555                     " EIP");
556         }
557
558         if (flags & DCMD_ADDRSPEC) {
559                 if (addr >= NCPU) {
560                         if (mdb_vread(&rec, sizeof (rec), addr) == -1) {
561                                 mdb_warn("couldn't read trap trace record "
562                                     "at %p", addr);
563                                 return (DCMD_ERR);
564                         }
565
566                         if (ttrace_walk(addr, &rec, &dcmd) == WALK_ERR)
567                                 return (DCMD_ERR);
568
569                         return (DCMD_OK);
570                 }
571                 dcmd.ttd_cpu = addr;
572         }
573
574         if (mdb_readvar(&use_apix, "apix_enable") == -1) {
575                 mdb_warn("failed to read apix_enable");
576                 use_apix = 0;
577         }
578
579         if (use_apix) {
580                 if (mdb_readvar(&d_apixs, "apixs") == -1) {
581                         mdb_warn("\nfailed to read apixs.");
582                         return (DCMD_ERR);
583                 }
584                 /* change to apix ttrace interrupt handler */
585                 ttrace_hdlr[4].t_hdlr = ttrace_apix_interrupt;
586         }
587
588         if (mdb_walk("ttrace", (mdb_walk_cb_t)ttrace_walk, &dcmd) == -1) {
589                 mdb_warn("couldn't walk 'ttrace'");
590                 return (DCMD_ERR);
591         }
592
593         return (DCMD_OK);
594 }
```

_____*unchanged_portion_omitted_*

```
752 static void
753 ptable_help(void)
754 {
755         mdb_printf(
756             "Given a PFN holding a page table, print its contents, and\n"
757             "the address of the corresponding htable structure.\n"
758             "\n"
759             "-m Interpret the PFN as an MFN (machine frame number)\n"
760             "-l force page table level (3 is top)\n");
761 }

763 static void
764 ptmap_help(void)
765 {
766         mdb_printf(
767             "Report all mappings represented by the page table hierarchy\n"
768             "rooted at the given cr3 value / physical address.\n"
769             "\n"
770             "-w run ::whatis on mapping start addresses\n");
771 }

773 /*
774  * NSEC_SHIFT is replicated here (it is not defined in a header file),
775  * but for amusement, the reader is directed to the comment that explains
776  * the rationale for this particular value on x86.  Spoiler:  the value is
777  * selected to accommodate 60 MHz Pentiums!  (And a confession:  if the voice
778  * in that comment sounds too familiar, it's because your author also wrote
779  * that code -- some fifteen years prior to this writing in 2011...)
780  */
781 #define NSEC_SHIFT 5

783 /*ARGSUSED*/
784 static int
785 scalehrtime_cmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
786 {
787         uint32_t nsec_scale;
788         hrtime_t tsc = addr, hrt;
789         unsigned int *tscp = (unsigned int *)&tsc;
790         uintptr_t scalehrtimef;
791         uint64_t scale;
792         GElf_Sym sym;

794         if (!(flags & DCMD_ADDRSPEC)) {
795                 if (argc != 1)
796                         return (DCMD_USAGE);

798                 switch (argv[0].a_type) {
799                 case MDB_TYPE_STRING:
800                         tsc = mdb_strtoull(argv[0].a_un.a_str);
801                         break;
802                 case MDB_TYPE_IMMEDIATE:
803                         tsc = argv[0].a_un.a_val;
804                         break;
805                 default:
806                         return (DCMD_USAGE);
807                 }
808         }

810         if (mdb_readsym(&scalehrtimef,
811             sizeof (scalehrtimef), "scalehrtimef") == -1) {
812                 mdb_warn("couldn't read 'scalehrtimef'");
813                 return (DCMD_ERR);
814         }

816         if (mdb_lookup_by_name("tsc_scalehrtime", &sym) == -1) {
```



```
817                 mdb_warn("couldn't find 'tsc_scalehrtime'");
818                 return (DCMD_ERR);
819         }

821         if (sym.st_value != scalehrtimef) {
822                 mdb_warn("::scalehrtime requires that scalehrtimef "
823                     "be set to tsc_scalehrtime\n");
824                 return (DCMD_ERR);
825         }

827         if (mdb_readsym(&nsec_scale, sizeof (nsec_scale), "nsec_scale") == -1) {
828                 mdb_warn("couldn't read 'nsec_scale'");
829                 return (DCMD_ERR);
830         }

832         scale = (uint64_t)nsec_scale;

834         hrt = ((uint64_t)tscp[1] * scale) << NSEC_SHIFT;
835         hrt += ((uint64_t)tscp[0] * scale) >> (32 - NSEC_SHIFT);

837         mdb_printf("0x%llx\n", hrt);

839         return (DCMD_OK);
840 }
```

*unchanged_portion_omitted_*

```
904 #ifdef _KMDB
905 /* ARGSUSED */
906 static int
907 crregs_dcmd(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
908 {
909         ulong_t cr0, cr2, cr3, cr4;
889         ulong_t cr0, cr4;
910         static const mdb_bitmask_t cr0_flag_bits[] = {
911                 { "PE",         CR0_PE,         CR0_PE },
912                 { "MP",         CR0_MP,         CR0_MP },
913                 { "EM",         CR0_EM,         CR0_EM },
914                 { "TS",         CR0_TS,         CR0_TS },
915                 { "ET",         CR0_ET,         CR0_ET },
916                 { "NE",         CR0_NE,         CR0_NE },
917                 { "WP",         CR0_WP,         CR0_WP },
918                 { "AM",         CR0_AM,         CR0_AM },
919                 { "NW",         CR0_NW,         CR0_NW },
920                 { "CD",         CR0_CD,         CR0_CD },
921                 { "PG",         CR0_PG,         CR0_PG },
922                 { NULL,         0,              0 }
923         };

925         static const mdb_bitmask_t cr3_flag_bits[] = {
926                 { "PCD",        CR3_PCD,        CR3_PCD },
927                 { "PWT",        CR3_PWT,        CR3_PWT },
928                 { NULL,         0,              0, }
929         };

931         static const mdb_bitmask_t cr4_flag_bits[] = {
932                 { "VME",        CR4_VME,        CR4_VME },
933                 { "PVI",        CR4_PVI,        CR4_PVI },
934                 { "TSD",        CR4_TSD,        CR4_TSD },
935                 { "DE",         CR4_DE,         CR4_DE },
936                 { "PSE",        CR4_PSE,        CR4_PSE },
937                 { "PAE",        CR4_PAE,        CR4_PAE },
938                 { "MCE",        CR4_MCE,        CR4_MCE },
939                 { "PGE",        CR4_PGE,        CR4_PGE },
940                 { "PCE",        CR4_PCE,        CR4_PCE },
941                 { "OSFXSR",     CR4_OSFXSR,     CR4_OSFXSR },
942                 { "OSXMMEXCPT", CR4_OSXMMEXCPT, CR4_OSXMMEXCPT },
```

```
                { "VMXE",       CR4_VMXE,       CR4_VMXE },
                { "SMXE",       CR4_SMXE,       CR4_SMXE },
                { "PCIDE",      CR4_PCIDE,      CR4_PCIDE },
                { "OSXSAVE",    CR4_OSXSAVE,    CR4_OSXSAVE },
                { "SMEP",       CR4_SMEP,       CR4_SMEP },
                { "SMAP",       CR4_SMAP,       CR4_SMAP },
                { NULL,         0,              0 }
        };

        cr0 = kmdb_unix_getcr0();
        cr2 = kmdb_unix_getcr2();
        cr3 = kmdb_unix_getcr3();
        cr4 = kmdb_unix_getcr4();
        mdb_printf("%%cr0 = 0x%lx <%b>\n", cr0, cr0, cr0_flag_bits);
        mdb_printf("%%cr2 = 0x%lx <%a>\n", cr2, cr2);

        if ((cr4 & CR4_PCIDE)) {
                mdb_printf("%%cr3 = 0x%lx <pfn:0x%lx pcid:%lu>\n", cr3,
                    cr3 >> MMU_PAGESHIFT, cr3 & MMU_PAGEOFFSET);
        } else {
                mdb_printf("%%cr3 = 0x%lx <pfn:0x%lx flags:%b>\n", cr3,
                    cr3 >> MMU_PAGESHIFT, cr3, cr3_flag_bits);
        }

        mdb_printf("%%cr4 = 0x%lx <%b>\n", cr4, cr4, cr4_flag_bits);

        mdb_printf("%%cr0 = 0x%08x <%b>\n", cr0, cr0, cr0_flag_bits);
        mdb_printf("%%cr4 = 0x%08x <%b>\n", cr4, cr4, cr4_flag_bits);
        return (DCMD_OK);
}
#endif

static const mdb_dcmd_t dcmds[] = {
        { "gate_desc", ":", "dump a gate descriptor", gate_desc },
        { "idt", ":[-v]", "dump an IDT", idt },
        { "ttrace", "[-x] [-t kthread]", "dump trap trace buffers", ttrace },
        { "ttrace", "[-x]", "dump trap trace buffers", ttrace },
        { "vatopfn", ":[-a as]", "translate address to physical page",
            va2pfn_dcmd },
        { "report_maps", ":[-m]",
            "Given PFN, report mappings / page table usage",
            report_maps_dcmd, report_maps_help },
        { "htables", "", "Given hat_t *, lists all its htable_t * values",
            htables_dcmd, htables_help },
        { "ptable", ":[-lm]", "Given PFN, dump contents of a page table",
        { "ptable", ":[-m]", "Given PFN, dump contents of a page table",
            ptable_dcmd, ptable_help },
        { "ptmap", ":", "Given a cr3 value, dump all mappings",
            ptmap_dcmd, ptmap_help },
        { "pte", ":[-l N]", "print human readable page table entry",
        { "pte", ":[-p XXXXX] [-l N]", "print human readable page table entry",
            pte_dcmd },
        { "pfntomfn", ":", "convert physical page to hypervisor machine page",
            pfntomfn_dcmd },
        { "mfntopfn", ":", "convert hypervisor machine page to physical page",
            mfntopfn_dcmd },
        { "memseg_list", ":", "show memseg list", memseg_list },
        { "scalehrtime", ":",
            "scale an unscaled high-res time", scalehrtime_cmd },
        { "x86_featureset", NULL, "dump the x86_featureset vector",
                x86_featureset_cmd },
#ifdef _KMDB
        { "crregs", NULL, "dump control registers", crregs_dcmd },
#endif
        { NULL }
};
```

_____***unchanged_portion_omitted_***

```
**********************************************************
     807 Fri Apr  6 17:24:56 2018
new/usr/src/cmd/mdb/i86pc/modules/unix/unix_sup.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
   1 /*
   2  * This file and its contents are supplied under the terms of the
   3  * Common Development and Distribution License ("CDDL"), version 1.0.
   4  * You may only use this file in accordance with the terms of version
   5  * 1.0 of the CDDL.
   6  *
   7  * A full copy of the text of the CDDL should have accompanied this
   8  * source.  A copy of the CDDL is also available via the Internet at
   9  * http://www.illumos.org/license/CDDL.
  10  */

  12 /*
  13  * Copyright 2018 Joyent, Inc.
  13  * Copyright 2015 Joyent, Inc.
  14  */

  16 #ifndef _UNIX_SUP_H
  17 #define _UNIX_SUP_H

  19 /*
  20  * Support routines for unix.
  21  */

  23 #include <sys/types.h>

  25 #ifdef __cplusplus
  26 extern "C" {
  27 #endif

  29 extern ulong_t kmdb_unix_getcr0(void);
  30 extern ulong_t kmdb_unix_getcr2(void);
  31 extern ulong_t kmdb_unix_getcr3(void);
  32 extern ulong_t kmdb_unix_getcr4(void);

  34 #ifdef __cplusplus
  35 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
    1500 Fri Apr  6 17:24:56 2018
new/usr/src/cmd/mdb/i86pc/modules/unix/unix_sup.s
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2018 Joyent, Inc.
 * Copyright 2015 Joyent, Inc.
 */

#if !defined(__lint)
        .file   "unix_sup.s"
#endif /* __lint */

/*
 * Support routines for the unix kmdb module
 */

#include <sys/asm_linkage.h>

#if defined(__lint)

#include <sys/types.h>

ulong_t
kmdb_unix_getcr0(void)
{ return (0); }

ulong_t
kmdb_unix_getcr3(void)
{ return (0); }

ulong_t
kmdb_unix_getcr4(void)
{ return (0); }

#else   /* __lint */

#if defined(__amd64)
        ENTRY(kmdb_unix_getcr0)
        movq %cr0, %rax
        ret
        SET_SIZE(kmdb_unix_getcr0)

        ENTRY(kmdb_unix_getcr2)
        movq %cr2, %rax
        ret
        SET_SIZE(kmdb_unix_getcr2)
```



```
        ENTRY(kmdb_unix_getcr3)
        movq %cr3, %rax
        ret
        SET_SIZE(kmdb_unix_getcr3)

        ENTRY(kmdb_unix_getcr4)
        movq %cr4, %rax
        ret
        SET_SIZE(kmdb_unix_getcr4)
______unchanged_portion_omitted_

        ENTRY(kmdb_unix_getcr2)
        movl %cr2, %eax
        ret
        SET_SIZE(kmdb_unix_getcr2)

        ENTRY(kmdb_unix_getcr3)
        movl %cr3, %eax
        ret
        SET_SIZE(kmdb_unix_getcr3)

        ENTRY(kmdb_unix_getcr4)
        movl %cr4, %eax
        ret
        SET_SIZE(kmdb_unix_getcr4)
______unchanged_portion_omitted_
```

```
**********************************************************
    1842 Fri Apr  6 17:24:56 2018
new/usr/src/cmd/mdb/intel/Makefile.kmdb
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
#
# 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.
#

PROMSRCS += \
        prom_env.c \
        prom_getchar.c \
        prom_init.c \
        prom_printf.c \
        prom_putchar.c

KMDBSRCS += \
        kaif.c \
        kmdb_dpi_isadep.c \
        kmdb_fault_isadep.c \
        kmdb_kdi_isadep.c \
        kmdb_promif_isadep.c \
        kvm_cpu_amd.c \
        kvm_cpu_p4.c \
        kvm_isadep.c

KMDBML += \
        kmdb_asmutil.s \
        kmdb_setcontext.s

KCTLSRCS += \
        kctl_isadep.c

CTXOFFUSERS = \
        kmdb_setcontext.o

CPPFLAGS += -DDIS_TEXT

$(CTXOFFUSERS) $(CTXOFFUSERS:%.o=%.ln): kmdb_context_off.h
```



```
kaif_activate.o kaif_activate.ln := CPPFLAGS += -D_MACHDEP -D_KMEMUSER

STANDLIBS += \
        ../libstandctf/libstandctf.so \
        $(SRC)/lib/libumem/$(MACHDIR)/libstandumem.so \
        ../libstand/libstand.a

KMDBLIBS = $(STANDLIBS) ../mdb_ks/kmod/mdb_ks

MAPFILE_SOURCES = \
        $(MAPFILE_SOURCES_COMMON) \
        ../../kmdb/kmdb_dpi_isadep.h \
        $(MAPFILE_SOURCES_$(MACH))

%.o: ../../../../../uts/intel/promif/%.c
        $(COMPILE.c) $<
        $(CTFCONVERT_O)

%.ln: ../../../../../uts/intel/promif/%.c
        $(LINT.c) -c $<
```

```
**********************************************************
    1053 Fri Apr  6 17:24:56 2018
new/usr/src/cmd/mdb/intel/ia32/Makefile.kmdb
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
#
# 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.
#ident  "%Z%%M% %I%     %E% SMI"
#

KMDBML += \
        kaif_invoke.s \
        kmdb_start.s

KMDBSRCS += \
        mdb_ia32util.c \
        kmdb_makecontext.c
        kmdb_makecontext.c \
        kvm_cpu_p6.c

SACPPFLAGS = -D__$(MACH)
```

```
**********************************************************
   19998 Fri Apr  6 17:24:56 2018
new/usr/src/cmd/mdb/intel/kmdb/kaif.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 debugger/"PROM" interface layer
 *
 * It makes more sense on SPARC. In reality, these interfaces deal with three
 * things: setting break/watchpoints, stepping, and interfacing with the KDI to
 * set up kmdb's IDT handlers.
 */

#include <kmdb/kmdb_dpi_impl.h>
#include <kmdb/kmdb_kdi.h>
#include <kmdb/kmdb_umemglue.h>
#include <kmdb/kaif.h>
#include <kmdb/kmdb_io.h>
#include <kmdb/kaif_start.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_isautil.h>
#include <mdb/mdb_io_impl.h>
#include <mdb/mdb_kreg_impl.h>
#include <mdb/mdb.h>

#include <sys/types.h>
#include <sys/bitmap.h>
#include <sys/termios.h>
#include <sys/kdi_impl.h>
```



```
#include <sys/sysmacros.h>

/*
 * This is the area containing the saved state when we enter
 * via kmdb's IDT entries.
 */
kdi_cpusave_t   *kaif_cpusave;
int             kaif_ncpusave;
kdi_drreg_t     kaif_drreg;

uint32_t        kaif_waptmap;

int             kaif_trap_switch;

void (*kaif_modchg_cb)(struct modctl *, int);

enum {
        M_SYSRET        = 0x07, /* after M_ESC */
        M_ESC           = 0x0f,
        M_SYSEXIT       = 0x35, /* after M_ESC */
        M_REX_LO        = 0x40, /* first REX prefix */
        M_REX_HI        = 0x4f, /* last REX prefix */
        M_PUSHF         = 0x9c, /* pushfl and pushfq */
        M_POPF          = 0x9d, /* popfl and popfq */
        M_INT3          = 0xcc,
        M_INTX          = 0xcd,
        M_INTO          = 0xce,
        M_IRET          = 0xcf,
        M_CLI           = 0xfa,
        M_STI           = 0xfb
};
_____unchanged_portion_omitted_

/*
 * Refuse to single-step or break within any stub that loads a user %cr3 value.
 * As the KDI traps are not careful to restore such a %cr3, this can all go
 * wrong, both spectacularly and subtly.
 */
static boolean_t
kaif_toxic_text(uintptr_t addr)
{
        static GElf_Sym toxic_syms[2] = { 0, };
        size_t i;

        if (toxic_syms[0].st_name == NULL) {
                if (mdb_tgt_lookup_by_name(mdb.m_target, MDB_TGT_OBJ_EXEC,
                    "tr_iret_user", &toxic_syms[0], NULL) != 0)
                        warn("couldn't find tr_iret_user\n");
                if (mdb_tgt_lookup_by_name(mdb.m_target, MDB_TGT_OBJ_EXEC,
                    "tr_mmu_flush_user_range", &toxic_syms[1], NULL) != 0)
                        warn("couldn't find tr_mmu_flush_user_range\n");
        }

        for (i = 0; i < ARRAY_SIZE(toxic_syms); i++) {
                if (addr >= toxic_syms[i].st_value &&
                    addr - toxic_syms[i].st_value < toxic_syms[i].st_size)
                        return (B_TRUE);
        }

        return (B_FALSE);
}

static int
kaif_brkpt_arm(uintptr_t addr, mdb_instr_t *instrp)
{
        mdb_instr_t bkpt = KAIF_BREAKPOINT_INSTR;
```

```
294         if (kaif_toxic_text(addr)) {
295                 warn("%a cannot be a breakpoint target\n", addr);
296                 return (set_errno(EMDB_TGTNOTSUP));
297         }

299         if (mdb_tgt_vread(mdb.m_target, instrp, sizeof (mdb_instr_t), addr) !=
300             sizeof (mdb_instr_t))
301                 return (-1); /* errno is set for us */

303         if (mdb_tgt_vwrite(mdb.m_target, &bkpt, sizeof (mdb_instr_t), addr) !=
304             sizeof (mdb_instr_t))
305                 return (-1); /* errno is set for us */

307         return (0);
308 }
_____unchanged_portion_omitted_

472 static int
473 kaif_step(void)
474 {
475         kreg_t pc, fl, oldfl, newfl, sp;
476         mdb_tgt_addr_t npc;
477         mdb_instr_t instr;
478         int emulated = 0, rchk = 0;
479         size_t pcoff = 0;

481         (void) kmdb_dpi_get_register("pc", &pc);

483         if (kaif_toxic_text(pc)) {
484                 warn("%a cannot be stepped\n", pc);
485                 return (set_errno(EMDB_TGTNOTSUP));
486         }

488         if ((npc = mdb_dis_nextins(mdb.m_disasm, mdb.m_target,
489             MDB_TGT_AS_VIRT, pc)) == pc) {
490                 warn("failed to decode instruction at %a for step\n", pc);
491                 return (set_errno(EINVAL));
492         }

494         /*
495          * Stepping behavior depends on the type of instruction.  It does not
496          * depend on the presence of a REX prefix, as the action we take for a
497          * given instruction doesn't currently vary for 32-bit instructions
498          * versus their 64-bit counterparts.
499          */
500         do {
501                 if (mdb_tgt_vread(mdb.m_target, &instr, sizeof (mdb_instr_t),
502                     pc + pcoff) != sizeof (mdb_instr_t)) {
503                         warn("failed to read at %p for step",
504                             (void *)(pc + pcoff));
505                         return (-1);
506                 }
507         } while (pcoff++, (instr >= M_REX_LO && instr <= M_REX_HI && !rchk++));

509         switch (instr) {
510         case M_IRET:
511                 warn("iret cannot be stepped\n");
512                 return (set_errno(EMDB_TGTNOTSUP));

514         case M_INT3:
515         case M_INTX:
516         case M_INTO:
517                 warn("int cannot be stepped\n");
518                 return (set_errno(EMDB_TGTNOTSUP));
```



```
520         case M_ESC:
521                 if (mdb_tgt_vread(mdb.m_target, &instr, sizeof (mdb_instr_t),
522                     pc + pcoff) != sizeof (mdb_instr_t)) {
523                         warn("failed to read at %p for step",
524                             (void *)(pc + pcoff));
525                         return (-1);
526                 }

528                 switch (instr) {
529                 case M_SYSRET:
530                         warn("sysret cannot be stepped\n");
531                         return (set_errno(EMDB_TGTNOTSUP));
532                 case M_SYSEXIT:
533                         warn("sysexit cannot be stepped\n");
534                         return (set_errno(EMDB_TGTNOTSUP));
535                 }
536                 break;

538         /*
539          * Some instructions need to be emulated.  We need to prevent direct
540          * manipulations of EFLAGS, so we'll emulate cli, sti.  pushfl and
541          * popfl also receive special handling, as they manipulate both EFLAGS
542          * and %esp.
543          */
544         case M_CLI:
545                 (void) kmdb_dpi_get_register(FLAGS_REG_NAME, &fl);
546                 fl &= ~KREG_EFLAGS_IF_MASK;
547                 (void) kmdb_dpi_set_register(FLAGS_REG_NAME, fl);

549                 emulated = 1;
550                 break;

552         case M_STI:
553                 (void) kmdb_dpi_get_register(FLAGS_REG_NAME, &fl);
554                 fl |= (1 << KREG_EFLAGS_IF_SHIFT);
555                 (void) kmdb_dpi_set_register(FLAGS_REG_NAME, fl);

557                 emulated = 1;
558                 break;

560         case M_POPF:
561                 /*
562                  * popfl will restore a pushed EFLAGS from the stack, and could
563                  * in so doing cause IF to be turned on, if only for a brief
564                  * period.  To avoid this, we'll secretly replace the stack's
565                  * EFLAGS with our decaffeinated brand.  We'll then manually
566                  * load our EFLAGS copy with the real verion after the step.
567                  */
568                 (void) kmdb_dpi_get_register("sp", &sp);
569                 (void) kmdb_dpi_get_register(FLAGS_REG_NAME, &fl);

571                 if (mdb_tgt_vread(mdb.m_target, &newfl, sizeof (kreg_t),
572                     sp) != sizeof (kreg_t)) {
573                         warn("failed to read " FLAGS_REG_NAME
574                             " at %p for popfl step\n", (void *)sp);
575                         return (set_errno(EMDB_TGTNOTSUP)); /* XXX ? */
576                 }

578                 fl = (fl & ~KREG_EFLAGS_IF_MASK) | KREG_EFLAGS_TF_MASK;

580                 if (mdb_tgt_vwrite(mdb.m_target, &fl, sizeof (kreg_t),
581                     sp) != sizeof (kreg_t)) {
582                         warn("failed to update " FLAGS_REG_NAME
583                             " at %p for popfl step\n", (void *)sp);
584                         return (set_errno(EMDB_TGTNOTSUP)); /* XXX ? */
585                 }
```

```
586                 break;
587         }
588
589         if (emulated) {
590                 (void) kmdb_dpi_set_register("pc", npc);
591                 return (0);
592         }
593
594         /* Do the step with IF off, and TF (step) on */
595         (void) kmdb_dpi_get_register(FLAGS_REG_NAME, &oldfl);
596         (void) kmdb_dpi_set_register(FLAGS_REG_NAME,
597             ((oldfl | (1 << KREG_EFLAGS_TF_SHIFT)) & ~KREG_EFLAGS_IF_MASK));
598
599         kmdb_dpi_resume_master(); /* ... there and back again ... */
600
601         /* EFLAGS has now changed, and may require tuning */
602
603         switch (instr) {
604         case M_POPF:
605                 /*
606                  * Use the EFLAGS we grabbed before the pop - see the pre-step
607                  * M_POPFL comment.
608                  */
609                 (void) kmdb_dpi_set_register(FLAGS_REG_NAME, newfl);
610                 return (0);
611
612         case M_PUSHF:
613                 /*
614                  * We pushed our modified EFLAGS (with IF and TF turned off)
615                  * onto the stack.  Replace the pushed version with our
616                  * unmodified one.
617                  */
618                 (void) kmdb_dpi_get_register("sp", &sp);
619
620                 if (mdb_tgt_vwrite(mdb.m_target, &oldfl, sizeof (kreg_t),
621                     sp) != sizeof (kreg_t)) {
622                         warn("failed to update pushed " FLAGS_REG_NAME
623                             " at %p after pushfl step\n", (void *)sp);
624                         return (set_errno(EMDB_TGTNOTSUP)); /* XXX ? */
625                 }
626
627                 /* Go back to using the EFLAGS we were using before the step */
628                 (void) kmdb_dpi_set_register(FLAGS_REG_NAME, oldfl);
629                 return (0);
630
631         default:
632                 /*
633                  * The stepped instruction may have altered EFLAGS.  We only
634                  * really care about the value of IF, and we know the stepped
635                  * instruction didn't alter it, so we can simply copy the
636                  * pre-step value.  We'll also need to turn TF back off.
637                  */
638                 (void) kmdb_dpi_get_register(FLAGS_REG_NAME, &fl);
639                 (void) kmdb_dpi_set_register(FLAGS_REG_NAME,
640                     ((fl & ~(KREG_EFLAGS_TF_MASK|KREG_EFLAGS_IF_MASK)) |
641                     (oldfl & KREG_EFLAGS_IF_MASK)));
642                 return (0);
643         }
644 }
645
646 /*
647  * The target has already configured the chip for branch step, leaving us to
648  * actually make the machine go.  Due to a number of issues involving
649  * the potential alteration of system state via instructions like sti, cli,
650  * pushfl, and popfl, we're going to treat this like a normal system resume.
651  * All CPUs will be released, on the kernel's IDT.  Our primary concern is
```



```
652  * the alteration/storage of our TF'd EFLAGS via pushfl and popfl.  There's no
653  * real workaround - we don't have opcode breakpoints - so the best we can do is
654  * to ensure that the world won't end if someone does bad things to EFLAGS.
655  *
656  * Two things can happen:
657  *  1. EFLAGS.TF may be cleared, either maliciously or via a popfl from saved
658  *     state.  The CPU will continue execution beyond the branch, and will not
659  *     reenter the debugger unless brought/sent in by other means.
660  *  2. Someone may pushlf the TF'd EFLAGS, and may stash a copy of it somewhere.
661  *     When the saved version is popfl'd back into place, the debugger will be
662  *     re-entered on a single-step trap.
663  */
664 static void
665 kaif_step_branch(void)
666 {
667         kreg_t fl;
668
669         (void) kmdb_dpi_get_register(FLAGS_REG_NAME, &fl);
670         (void) kmdb_dpi_set_register(FLAGS_REG_NAME,
671             (fl | (1 << KREG_EFLAGS_TF_SHIFT)));
672
673         kmdb_dpi_resume_master();
674
675         (void) kmdb_dpi_set_register(FLAGS_REG_NAME, fl);
676 }
677
678 /*ARGSUSED*/
679 static uintptr_t
680 kaif_call(uintptr_t funcva, uint_t argc, const uintptr_t argv[])
681 {
682         return (kaif_invoke(funcva, argc, argv));
683 }
_____unchanged_portion_omitted_
759 static void
760 kaif_msr_add(const kdi_msr_t *msrs)
761 {
762         kdi_msr_t *save;
763         size_t nr_msrs = 0;
764         size_t i;
765
766         while (msrs[nr_msrs].msr_num != 0)
767                 nr_msrs++;
768         /* we want to copy the terminating kdi_msr_t too */
769         nr_msrs++;
770
771         save = mdb_zalloc(sizeof (kdi_msr_t) * nr_msrs * kaif_ncpusave,
772             UM_SLEEP);
773
774         for (i = 0; i < kaif_ncpusave; i++)
775                 bcopy(msrs, &save[nr_msrs * i], sizeof (kdi_msr_t) * nr_msrs);
776
777         kmdb_kdi_set_debug_msrs(save);
778 }
779
780 static uint64_t
781 kaif_msr_get(int cpuid, uint_t num)
782 {
783         kdi_cpusave_t *save;
784         kdi_msr_t *msr;
785         int i;
786
787         if ((save = kaif_cpuid2save(cpuid)) == NULL)
788                 return (-1); /* errno is set for us */
789
790         msr = save->krs_msr;
```

```
760         for (i = 0; msr[i].msr_num != 0; i++) {
761                 if (msr[i].msr_num == num && (msr[i].msr_type & KDI_MSR_READ))
762                         return (msr[i].kdi_msr_val);
763         }

765         return (0);
766 }

735 void
736 kaif_trap_set_debugger(void)
737 {
738         kmdb_kdi_idt_switch(NULL);
739 }
```

unchanged_portion_omitted_

```
833 dpi_ops_t kmdb_dpi_ops = {
834         kaif_init,
835         kaif_activate,
836         kmdb_kdi_deactivate,
837         kaif_enter_mon,
838         kaif_modchg_register,
839         kaif_modchg_cancel,
840         kaif_get_cpu_state,
841         kaif_get_master_cpuid,
842         kaif_get_gregs,
843         kaif_get_register,
844         kaif_set_register,
845         kaif_brkpt_arm,
846         kaif_brkpt_disarm,
847         kaif_wapt_validate,
848         kaif_wapt_reserve,
849         kaif_wapt_release,
850         kaif_wapt_arm,
851         kaif_wapt_disarm,
852         kaif_wapt_match,
853         kaif_step,
887         kaif_step_branch,
854         kaif_call,
855         kaif_dump_crumbs,
890         kaif_msr_add,
891         kaif_msr_get,
856 };
```

unchanged_portion_omitted_

```
**********************************************************
    3622 Fri Apr  6 17:24:57 2018
new/usr/src/cmd/mdb/intel/kmdb/kmdb_dpi_isadep.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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"

/*
 * Intel-specific portions of the DPI
 */

#include <sys/types.h>
#include <sys/trap.h>

#include <kmdb/kmdb_dpi_impl.h>
#include <kmdb/kmdb_fault.h>
#include <kmdb/kmdb_kdi.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_kreg.h>
#include <mdb/mdb.h>

void
kmdb_dpi_handle_fault(kreg_t trapno, kreg_t pc, kreg_t sp, int cpuid)
{
        kmdb_kdi_system_claim();

        mdb_dprintf(MDB_DBG_DPI, "\ndpi_handle_fault: trapno %u, pc 0x%0?p, "
            "sp 0x%0?p\n", (int)trapno, pc, sp);

        switch (trapno) {
        case T_GPFLT:
                errno = EACCES;
        default:
                errno = EMDB_NOMAP;
        }

        if (kmdb_dpi_fault_pcb != NULL) {
                longjmp(*kmdb_dpi_fault_pcb, 1);
                /*NOTREACHED*/
        }

        /* Debugger fault */
        kmdb_fault(trapno, pc, sp, cpuid);
}
```



```
_____unchanged_portion_omitted_

void
kmdb_dpi_reboot(void)
{
        /*
         * We're going to skip all of the niceties we employ in resume_common,
         * as we don't plan to ever return.
         */
        longjmp(kmdb_dpi_entry_pcb, KMDB_DPI_CMD_REBOOT);
}

void
kmdb_dpi_msr_add(const kdi_msr_t *msrs)
{
        mdb.m_dpi->dpo_msr_add(msrs);
}

uint64_t
kmdb_dpi_msr_get(uint_t msr)
{
        return (mdb.m_dpi->dpo_msr_get(DPI_MASTER_CPUID, msr));
}

uint64_t
kmdb_dpi_msr_get_by_cpu(int cpuid, uint_t msr)
{
        return (mdb.m_dpi->dpo_msr_get(cpuid, msr));
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    1342 Fri Apr  6 17:24:57 2018
new/usr/src/cmd/mdb/intel/kmdb/kmdb_dpi_isadep.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 _KMDB_DPI_ISADEP_H
#define _KMDB_DPI_ISADEP_H

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

#ifndef _ASM
#include <mdb/mdb_isautil.h>
#include <sys/kdi_machimpl.h>
#endif

#ifdef __cplusplus
extern "C" {
#endif

#ifndef _ASM

extern void kmdb_dpi_handle_fault(kreg_t, kreg_t, kreg_t, int);

extern void kmdb_dpi_reboot(void) __NORETURN;

extern void kmdb_dpi_msr_add(const kdi_msr_t *);
extern uint64_t kmdb_dpi_msr_get(uint_t);
extern uint64_t kmdb_dpi_msr_get_by_cpu(int, uint_t);

#endif /* _ASM */

#ifdef __cplusplus
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    2397 Fri Apr  6 17:24:57 2018
new/usr/src/cmd/mdb/intel/kmdb/kmdb_kdi_isadep.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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"

#include <sys/types.h>
#include <sys/kdi_impl.h>
#include <sys/segments.h>
#include <sys/cpuvar.h>

#include <mdb/mdb_debug.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_umem.h>
#include <kmdb/kmdb_dpi.h>
#include <mdb/mdb.h>

/*ARGSUSED*/
void
kmdb_kdi_stop_slaves(int my_cpuid, int doxc)
{
        /* Stop other CPUs if there are CPUs to stop */
        mdb.m_kdi->mkdi_stop_slaves(my_cpuid, doxc);
}
```

_____*unchanged_portion_omitted_*

```
void
kmdb_kdi_set_debug_msrs(kdi_msr_t *msrs)
{
        mdb.m_kdi->mkdi_set_debug_msrs(msrs);
}
```

```
void
kmdb_kdi_memrange_add(caddr_t base, size_t len)
```



```
{
        mdb.m_kdi->mkdi_memrange_add(base, len);
}
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    1627 Fri Apr  6 17:24:57 2018
new/usr/src/cmd/mdb/intel/kmdb/kmdb_kdi_isadep.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 _KMDB_KDI_ISADEP_H
#define _KMDB_KDI_ISADEP_H

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

#include <sys/types.h>
#include <sys/kdi_machimpl.h>

#include <mdb/mdb_target.h>

#ifdef __cplusplus
extern "C" {
#endif

struct gate_desc;

extern void kmdb_kdi_activate(kdi_main_t, kdi_cpusave_t *, int);
extern void kmdb_kdi_deactivate(void);

extern void kmdb_kdi_idt_switch(kdi_cpusave_t *);

extern void kmdb_kdi_update_drreg(kdi_drreg_t *);
extern void kmdb_kdi_set_debug_msrs(kdi_msr_t *);

extern uintptr_t kmdb_kdi_get_userlimit(void);

extern int kmdb_kdi_get_cpuinfo(uint_t *, uint_t *, uint_t *);

extern void kmdb_kdi_memrange_add(caddr_t, size_t);
```



```
extern void kmdb_kdi_reboot(void);

#ifdef __cplusplus
}
#endif
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   14222 Fri Apr  6 17:24:57 2018
new/usr/src/cmd/mdb/intel/kmdb/kvm_isadep.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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"

/*
 * isa-dependent portions of the kmdb target
 */

#include <kmdb/kvm.h>
#include <kmdb/kvm_cpu.h>
#include <kmdb/kmdb_kdi.h>
#include <kmdb/kmdb_asmutil.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_list.h>
#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_isautil.h>
#include <mdb/mdb_kreg_impl.h>
#include <mdb/mdb.h>

#include <sys/types.h>
#include <sys/frame.h>
#include <sys/trap.h>
#include <sys/bitmap.h>
#include <sys/pci_impl.h>

/* Higher than the highest trap number for which we have a defined specifier */
#define KMT_MAXTRAPNO   0x20

#define IOPORTLIMIT     0xffff  /* XXX find a new home for this */

const char *
```



```
kmt_def_dismode(void)
{
#ifdef  __amd64
        return ("amd64");
#else
        return ("ia32");
#endif
}
```

_____*unchanged_portion_omitted_*

```
int
kmt_step_branch(mdb_tgt_t *t)
{
        kmt_data_t *kmt = t->t_data;

        return (kmt_cpu_step_branch(t, kmt->kmt_cpu));
}

/*
 * Return the address of the next instruction following a call, or return -1
 * and set errno to EAGAIN if the target should just single-step.
 */
int
kmt_next(mdb_tgt_t *t, uintptr_t *p)
{
        kreg_t pc;
        mdb_instr_t instr;

        (void) kmdb_dpi_get_register("pc", &pc);

        if (mdb_tgt_vread(t, &instr, sizeof (mdb_instr_t), pc) !=
            sizeof (mdb_instr_t))
                return (-1); /* errno is set for us */

        return (mdb_isa_next(t, p, pc, instr));
}
```

_____*unchanged_portion_omitted_*

```
int
kmt_msr_validate(const kdi_msr_t *msr)
{
        uint64_t val;

        for (/* */; msr->msr_num != 0; msr++) {
                if (kmt_rwmsr(msr->msr_num, &val, rdmsr) < 0)
                        return (0);
        }

        return (1);
}

/*ARGSUSED*/
ssize_t
kmt_write(mdb_tgt_t *t, const void *buf, size_t nbytes, uintptr_t addr)
{
        if (!(t->t_flags & MDB_TGT_F_ALLOWIO) &&
            (nbytes = kmdb_kdi_range_is_nontoxic(addr, nbytes, 1)) == 0)
                return (set_errno(EMDB_NOMAP));

        /*
         * No writes to user space are allowed.  If we were to allow it, we'd
         * be in the unfortunate situation where kmdb could place a breakpoint
         * on a userspace executable page; this dirty page would end up being
         * flushed back to disk, incurring sadness when it's next executed.
         * Besides, we can't allow trapping in from userspace anyway.
         */
```

```
 365         if (addr < kmdb_kdi_get_userlimit())
 366                 return (set_errno(EMDB_TGTNOTSUP));

 368         return (kmt_rw(t, (void *)buf, nbytes, addr, kmt_writer));
 369 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1670 Fri Apr  6 17:24:58 2018
new/usr/src/cmd/mdb/intel/kmdb/kvm_isadep.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 _KVM_ISADEP_H
#define _KVM_ISADEP_H

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

#ifdef __cplusplus
extern "C" {
#endif

extern uintptr_t kmt_invoke(uintptr_t, uint_t, const uintptr_t *);

extern void kmt_in(void *, size_t, uintptr_t);
extern void kmt_out(void *, size_t, uintptr_t);

extern int kmt_in_dcmd(uintptr_t, uint_t, int, const mdb_arg_t *);
extern int kmt_out_dcmd(uintptr_t, uint_t, int, const mdb_arg_t *);

extern int kmt_rdmsr(uintptr_t, uint_t, int, const mdb_arg_t *);
extern int kmt_wrmsr(uintptr_t, uint_t, int, const mdb_arg_t *);

extern int kmt_rdpcicfg(uintptr_t, uint_t, int, const mdb_arg_t *);
extern int kmt_wrpcicfg(uintptr_t, uint_t, int, const mdb_arg_t *);
extern int kmt_msr_validate(const kdi_msr_t *);

#ifdef __cplusplus
}
```

_____unchanged_portion_omitted_

```
**********************************************************
    9127 Fri Apr  6 17:24:58 2018
new/usr/src/cmd/mdb/intel/mdb/kvm_amd64dep.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

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

/*
 * Libkvm Kernel Target Intel 64-bit component
 *
 * This file provides the ISA-dependent portion of the libkvm kernel target.
 * For more details on the implementation refer to mdb_kvm.c.
 */

#include <sys/types.h>
#include <sys/reg.h>
#include <sys/frame.h>
#include <sys/stack.h>
#include <sys/sysmacros.h>
#include <sys/panic.h>
#include <sys/privregs.h>
#include <strings.h>

#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_disasm.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_conf.h>
#include <mdb/mdb_kreg_impl.h>
#include <mdb/mdb_amd64util.h>
#include <mdb/kvm_isadep.h>
#include <mdb/mdb_kvm.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb.h>
```



```
/*ARGSUSED*/
int
kt_regs(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
{
        mdb_amd64_printregs((const mdb_tgt_gregset_t *)addr);
        return (DCMD_OK);
}
______unchanged_portion_omitted_

const mdb_tgt_ops_t kt_amd64_ops = {
        kt_setflags,                            /* t_setflags */
        kt_setcontext,                          /* t_setcontext */
        kt_activate,                            /* t_activate */
        kt_deactivate,                          /* t_deactivate */
        (void (*)()) mdb_tgt_nop,               /* t_periodic */
        kt_destroy,                             /* t_destroy */
        kt_name,                                /* t_name */
        (const char *(*)()) mdb_conf_isa,       /* t_isa */
        kt_platform,                            /* t_platform */
        kt_uname,                               /* t_uname */
        kt_dmodel,                              /* t_dmodel */
        kt_aread,                               /* t_aread */
        kt_awrite,                              /* t_awrite */
        kt_vread,                               /* t_vread */
        kt_vwrite,                              /* t_vwrite */
        kt_pread,                               /* t_pread */
        kt_pwrite,                              /* t_pwrite */
        kt_fread,                               /* t_fread */
        kt_fwrite,                              /* t_fwrite */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_ioread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_iowrite */
        kt_vtop,                                /* t_vtop */
        kt_lookup_by_name,                      /* t_lookup_by_name */
        kt_lookup_by_addr,                      /* t_lookup_by_addr */
        kt_symbol_iter,                         /* t_symbol_iter */
        kt_mapping_iter,                        /* t_mapping_iter */
        kt_object_iter,                         /* t_object_iter */
        kt_addr_to_map,                         /* t_addr_to_map */
        kt_name_to_map,                         /* t_name_to_map */
        kt_addr_to_ctf,                         /* t_addr_to_ctf */
        kt_name_to_ctf,                         /* t_name_to_ctf */
        kt_status,                              /* t_status */
        (int (*)()) mdb_tgt_notsup,             /* t_run */
        (int (*)()) mdb_tgt_notsup,             /* t_step */
        (int (*)()) mdb_tgt_notsup,             /* t_step_out */
        (int (*)()) mdb_tgt_notsup,             /* t_step_branch */
        (int (*)()) mdb_tgt_notsup,             /* t_next */
        (int (*)()) mdb_tgt_notsup,             /* t_cont */
        (int (*)()) mdb_tgt_notsup,             /* t_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_vbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_sbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_pwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_vwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_iowapt */
        (int (*)()) mdb_tgt_null,               /* t_add_sysenter */
        (int (*)()) mdb_tgt_null,               /* t_add_sysexit */
        (int (*)()) mdb_tgt_null,               /* t_add_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_fault */
        kt_getareg,                             /* t_getareg */
        kt_putareg,                             /* t_putareg */
        mdb_amd64_kvm_stack_iter,               /* t_stack_iter */
        (int (*)()) mdb_tgt_notsup              /* t_auxv */
};
______unchanged_portion_omitted_
```

```
**********************************************************
    9471 Fri Apr  6 17:24:58 2018
new/usr/src/cmd/mdb/intel/mdb/kvm_ia32dep.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

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

/*
 * Libkvm Kernel Target Intel 32-bit component
 *
 * This file provides the ISA-dependent portion of the libkvm kernel target.
 * For more details on the implementation refer to mdb_kvm.c.
 */

#include <sys/types.h>
#include <sys/regset.h>
#include <sys/frame.h>
#include <sys/stack.h>
#include <sys/sysmacros.h>
#include <sys/panic.h>
#include <strings.h>

#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_disasm.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_conf.h>
#include <mdb/mdb_kreg_impl.h>
#include <mdb/mdb_ia32util.h>
#include <mdb/kvm_isadep.h>
#include <mdb/mdb_kvm.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb.h>
```



```
/*ARGSUSED*/
int
kt_regs(uintptr_t addr, uint_t flags, int argc, const mdb_arg_t *argv)
{
        mdb_ia32_printregs((const mdb_tgt_gregset_t *)addr);
        return (DCMD_OK);
}
______unchanged_portion_omitted_

const mdb_tgt_ops_t kt_ia32_ops = {
        kt_setflags,                            /* t_setflags */
        kt_setcontext,                          /* t_setcontext */
        kt_activate,                            /* t_activate */
        kt_deactivate,                          /* t_deactivate */
        (void (*)()) mdb_tgt_nop,               /* t_periodic */
        kt_destroy,                             /* t_destroy */
        kt_name,                                /* t_name */
        (const char *(*)()) mdb_conf_isa,       /* t_isa */
        kt_platform,                            /* t_platform */
        kt_uname,                               /* t_uname */
        kt_dmodel,                              /* t_dmodel */
        kt_aread,                               /* t_aread */
        kt_awrite,                              /* t_awrite */
        kt_vread,                               /* t_vread */
        kt_vwrite,                              /* t_vwrite */
        kt_pread,                               /* t_pread */
        kt_pwrite,                              /* t_pwrite */
        kt_fread,                               /* t_fread */
        kt_fwrite,                              /* t_fwrite */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_ioread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_iowrite */
        kt_vtop,                                /* t_vtop */
        kt_lookup_by_name,                      /* t_lookup_by_name */
        kt_lookup_by_addr,                      /* t_lookup_by_addr */
        kt_symbol_iter,                         /* t_symbol_iter */
        kt_mapping_iter,                        /* t_mapping_iter */
        kt_object_iter,                         /* t_object_iter */
        kt_addr_to_map,                         /* t_addr_to_map */
        kt_name_to_map,                         /* t_name_to_map */
        kt_addr_to_ctf,                         /* t_addr_to_ctf */
        kt_name_to_ctf,                         /* t_name_to_ctf */
        kt_status,                              /* t_status */
        (int (*)()) mdb_tgt_notsup,             /* t_run */
        (int (*)()) mdb_tgt_notsup,             /* t_step */
        (int (*)()) mdb_tgt_notsup,             /* t_step_out */
        (int (*)()) mdb_tgt_notsup,             /* t_step_branch */
        (int (*)()) mdb_tgt_notsup,             /* t_next */
        (int (*)()) mdb_tgt_notsup,             /* t_cont */
        (int (*)()) mdb_tgt_notsup,             /* t_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_vbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_sbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_pwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_vwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_iowapt */
        (int (*)()) mdb_tgt_null,               /* t_add_sysenter */
        (int (*)()) mdb_tgt_null,               /* t_add_sysexit */
        (int (*)()) mdb_tgt_null,               /* t_add_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_fault */
        kt_getareg,                             /* t_getareg */
        kt_putareg,                             /* t_putareg */
        mdb_ia32_kvm_stack_iter,                /* t_stack_iter */
        (int (*)()) mdb_tgt_notsup              /* t_auxv */
};
______unchanged_portion_omitted_
```

```
**********************************************************
   16759 Fri Apr  6 17:24:58 2018
new/usr/src/cmd/mdb/intel/mdb/mdb_amd64util.c
9441 kmdb should stash %cr3 in kdiregs
Reviewed by: John Levon <john.levon@joyent.com>
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2018, Joyent, Inc.  All rights reserved.
 * Copyright (c) 2012, Joyent, Inc.  All rights reserved.
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 2013 by Delphix. All rights reserved.
 */

#include <sys/types.h>
#include <sys/reg.h>
#include <sys/privregs.h>
#include <sys/stack.h>
#include <sys/frame.h>

#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_kreg_impl.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_amd64util.h>
#include <mdb/mdb_ctf.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb.h>

#include <saveargs.h>

/*
 * This array is used by the getareg and putareg entry points, and also by our
 * register variable discipline.
 */

const mdb_tgt_regdesc_t mdb_amd64_kregs[] = {
        { "savfp", KREG_SAVFP, MDB_TGT_R_EXPORT },
        { "savpc", KREG_SAVPC, MDB_TGT_R_EXPORT },
```



```
        { "rdi", KREG_RDI, MDB_TGT_R_EXPORT },
        { "edi", KREG_RDI, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "di",  KREG_RDI, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "dil", KREG_RDI, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "rsi", KREG_RSI, MDB_TGT_R_EXPORT },
        { "esi", KREG_RSI, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "si",  KREG_RSI, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "sil", KREG_RSI, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "rdx", KREG_RDX, MDB_TGT_R_EXPORT },
        { "edx", KREG_RDX, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "dx",  KREG_RDX, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "dh",  KREG_RDX, MDB_TGT_R_EXPORT | MDB_TGT_R_8H },
        { "dl",  KREG_RDX, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "rcx", KREG_RCX, MDB_TGT_R_EXPORT },
        { "ecx", KREG_RCX, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "cx",  KREG_RCX, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "ch",  KREG_RCX, MDB_TGT_R_EXPORT | MDB_TGT_R_8H },
        { "cl",  KREG_RCX, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "r8", KREG_R8, MDB_TGT_R_EXPORT },
        { "r8d", KREG_R8,  MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "r8w", KREG_R8,  MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "r8l", KREG_R8,  MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "r9", KREG_R9, MDB_TGT_R_EXPORT },
        { "r9d", KREG_R8,  MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "r9w", KREG_R8,  MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "r9l", KREG_R8,  MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "rax", KREG_RAX, MDB_TGT_R_EXPORT },
        { "eax", KREG_RAX, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "ax",  KREG_RAX, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "ah",  KREG_RAX, MDB_TGT_R_EXPORT | MDB_TGT_R_8H },
        { "al",  KREG_RAX, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "rbx", KREG_RBX, MDB_TGT_R_EXPORT },
        { "ebx", KREG_RBX, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "bx",  KREG_RBX, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "bh",  KREG_RBX, MDB_TGT_R_EXPORT | MDB_TGT_R_8H },
        { "bl",  KREG_RBX, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "rbp", KREG_RBP, MDB_TGT_R_EXPORT },
        { "ebp", KREG_RBP, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "bp",  KREG_RBP, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "bpl", KREG_RBP, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "r10", KREG_R10, MDB_TGT_R_EXPORT },
        { "r10d", KREG_R10, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "r10w", KREG_R10, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "r10l", KREG_R10, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "r11", KREG_R11, MDB_TGT_R_EXPORT },
        { "r11d", KREG_R11, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "r11w", KREG_R11, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "r11l", KREG_R11, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "r12", KREG_R12, MDB_TGT_R_EXPORT },
        { "r12d", KREG_R12, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "r12w", KREG_R12, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "r12l", KREG_R12, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "r13", KREG_R13, MDB_TGT_R_EXPORT },
        { "r13d", KREG_R13, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "r13w", KREG_R13, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "r13l", KREG_R13, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "r14", KREG_R14, MDB_TGT_R_EXPORT },
        { "r14d", KREG_R14, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "r14w", KREG_R14, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "r14l", KREG_R14, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "r15", KREG_R15, MDB_TGT_R_EXPORT },
        { "r15d", KREG_R15, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
        { "r15w", KREG_R15, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
        { "r15l", KREG_R15, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
        { "ds", KREG_DS, MDB_TGT_R_EXPORT },
        { "es", KREG_ES, MDB_TGT_R_EXPORT },
```

```
123         { "fs", KREG_FS, MDB_TGT_R_EXPORT },
124         { "gs", KREG_GS, MDB_TGT_R_EXPORT },
125         { "trapno", KREG_TRAPNO, MDB_TGT_R_EXPORT | MDB_TGT_R_PRIV },
126         { "err", KREG_ERR, MDB_TGT_R_EXPORT | MDB_TGT_R_PRIV },
127         { "rip", KREG_RIP, MDB_TGT_R_EXPORT },
128         { "cs", KREG_CS, MDB_TGT_R_EXPORT },
129         { "rflags", KREG_RFLAGS, MDB_TGT_R_EXPORT },
130         { "eflags", KREG_RFLAGS, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
131         { "rsp", KREG_RSP, MDB_TGT_R_EXPORT },
132         { "esp", KREG_RSP, MDB_TGT_R_EXPORT | MDB_TGT_R_32 },
133         { "sp",  KREG_RSP, MDB_TGT_R_EXPORT | MDB_TGT_R_16 },
134         { "spl", KREG_RSP, MDB_TGT_R_EXPORT | MDB_TGT_R_8L },
135         { "ss", KREG_SS, MDB_TGT_R_EXPORT },
136         { "gsbase", KREG_GSBASE, MDB_TGT_R_EXPORT },
137         { "kgsbase", KREG_KGSBASE, MDB_TGT_R_EXPORT },
138         { "cr2", KREG_CR2, MDB_TGT_R_EXPORT },
139         { "cr3", KREG_CR3, MDB_TGT_R_EXPORT },
140         { NULL, 0, 0 }
141 };

143 void
144 mdb_amd64_printregs(const mdb_tgt_gregset_t *gregs)
145 {
146         const kreg_t *kregs = &gregs->kregs[0];
147         kreg_t rflags = kregs[KREG_RFLAGS];

149 #define GETREG2(x) ((uintptr_t)kregs[(x)]), ((uintptr_t)kregs[(x)])

151         mdb_printf("%%rax = 0x%0?p %15A %%r9  = 0x%0?p %A\n",
152             GETREG2(KREG_RAX), GETREG2(KREG_R9));
153         mdb_printf("%%rbx = 0x%0?p %15A %%r10 = 0x%0?p %A\n",
154             GETREG2(KREG_RBX), GETREG2(KREG_R10));
155         mdb_printf("%%rcx = 0x%0?p %15A %%r11 = 0x%0?p %A\n",
156             GETREG2(KREG_RCX), GETREG2(KREG_R11));
157         mdb_printf("%%rdx = 0x%0?p %15A %%r12 = 0x%0?p %A\n",
158             GETREG2(KREG_RDX), GETREG2(KREG_R12));
159         mdb_printf("%%rsi = 0x%0?p %15A %%r13 = 0x%0?p %A\n",
160             GETREG2(KREG_RSI), GETREG2(KREG_R13));
161         mdb_printf("%%rdi = 0x%0?p %15A %%r14 = 0x%0?p %A\n",
162             GETREG2(KREG_RDI), GETREG2(KREG_R14));
163         mdb_printf("%%r8  = 0x%0?p %15A %%r15 = 0x%0?p %A\n\n",
164             GETREG2(KREG_R8), GETREG2(KREG_R15));

166         mdb_printf("%%rip = 0x%0?p %A\n", GETREG2(KREG_RIP));
167         mdb_printf("%%rbp = 0x%0?p\n", kregs[KREG_RBP]);
168         mdb_printf("%%rsp = 0x%0?p\n", kregs[KREG_RSP]);

170         mdb_printf("%%rflags = 0x%08x\n", rflags);

172         mdb_printf("  id=%u vip=%u vif=%u ac=%u vm=%u rf=%u nt=%u iopl=0x%x\n",
173             (rflags & KREG_EFLAGS_ID_MASK) >> KREG_EFLAGS_ID_SHIFT,
174             (rflags & KREG_EFLAGS_VIP_MASK) >> KREG_EFLAGS_VIP_SHIFT,
175             (rflags & KREG_EFLAGS_VIF_MASK) >> KREG_EFLAGS_VIF_SHIFT,
176             (rflags & KREG_EFLAGS_AC_MASK) >> KREG_EFLAGS_AC_SHIFT,
177             (rflags & KREG_EFLAGS_VM_MASK) >> KREG_EFLAGS_VM_SHIFT,
178             (rflags & KREG_EFLAGS_RF_MASK) >> KREG_EFLAGS_RF_SHIFT,
179             (rflags & KREG_EFLAGS_NT_MASK) >> KREG_EFLAGS_NT_SHIFT,
180             (rflags & KREG_EFLAGS_IOPL_MASK) >> KREG_EFLAGS_IOPL_SHIFT);

182         mdb_printf("  status=<%s,%s,%s,%s,%s,%s,%s,%s,%s>\n\n",
183             (rflags & KREG_EFLAGS_OF_MASK) ? "OF" : "of",
184             (rflags & KREG_EFLAGS_DF_MASK) ? "DF" : "df",
185             (rflags & KREG_EFLAGS_IF_MASK) ? "IF" : "if",
186             (rflags & KREG_EFLAGS_TF_MASK) ? "TF" : "tf",
187             (rflags & KREG_EFLAGS_SF_MASK) ? "SF" : "sf",
188             (rflags & KREG_EFLAGS_ZF_MASK) ? "ZF" : "zf",
```



```
189             (rflags & KREG_EFLAGS_AF_MASK) ? "AF" : "af",
190             (rflags & KREG_EFLAGS_PF_MASK) ? "PF" : "pf",
191             (rflags & KREG_EFLAGS_CF_MASK) ? "CF" : "cf");

193         mdb_printf("%%cs = 0x%04x\t%%ds = 0x%04x\t"
194             "%%es = 0x%04x\t%%fs = 0x%04x\n", kregs[KREG_CS], kregs[KREG_DS],
195             kregs[KREG_ES], kregs[KREG_FS] & 0xffff);
196         mdb_printf("%%gs = 0x%04x\t%%gsbase = 0x%lx\t%%kgsbase = 0x%lx\n",
197             kregs[KREG_GS] & 0xffff, kregs[KREG_GSBASE], kregs[KREG_KGSBASE]);
198         mdb_printf("%%trapno = 0x%x\t%%err = 0x%x\t%%cr2 = 0x%lx\t"
199             "%%cr3 = 0x%lx\n", kregs[KREG_TRAPNO], kregs[KREG_ERR],
200             kregs[KREG_CR2], kregs[KREG_CR3]);
189         mdb_printf("%24s%%cs = 0x%04x\t%%ds = 0x%04x\t%%es = 0x%04x\n",
190             " ", kregs[KREG_CS], kregs[KREG_DS], kregs[KREG_ES]);

192         mdb_printf("%%trapno = 0x%x\t\t%%fs = 0x%04x\t%%gs = 0x%04x\n",
193             kregs[KREG_TRAPNO], (kregs[KREG_FS] & 0xffff),
194             (kregs[KREG_GS] & 0xffff));
195         mdb_printf("   %%err = 0x%x\n", kregs[KREG_ERR]);
201 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5339 Fri Apr  6 17:24:58 2018
new/usr/src/cmd/mdb/intel/mdb/mdb_kreg.h
9441 kmdb should stash %cr3 in kdiregs
Reviewed by: John Levon <john.levon@joyent.com>
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 _MDB_KREG_H
#define _MDB_KREG_H

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

#include <sys/kdi_regs.h>
#ifndef _ASM
#include <sys/types.h>
#endif

#ifdef  __cplusplus
extern "C" {
#endif

#ifndef _ASM
#ifdef __amd64
typedef uint64_t kreg_t;
#else   /* __amd64 */
typedef uint32_t kreg_t;
#endif  /* __amd64 */
#endif  /* !_ASM */

#define KREG_NGREG      KDIREG_NGREG

/*
 * The order of these registers corresponds to a slightly altered struct regs,
 * in the order kmdb entry pushes onto the stack.
 */

#ifdef __amd64
```



```
#define KREG_SAVFP      KDIREG_SAVFP
#define KREG_SAVPC      KDIREG_SAVPC
#define KREG_RDI        KDIREG_RDI
#define KREG_RSI        KDIREG_RSI
#define KREG_RDX        KDIREG_RDX
#define KREG_RCX        KDIREG_RCX
#define KREG_R8         KDIREG_R8
#define KREG_R9         KDIREG_R9
#define KREG_RAX        KDIREG_RAX
#define KREG_RBX        KDIREG_RBX
#define KREG_RBP        KDIREG_RBP
#define KREG_R10        KDIREG_R10
#define KREG_R11        KDIREG_R11
#define KREG_R12        KDIREG_R12
#define KREG_R13        KDIREG_R13
#define KREG_R14        KDIREG_R14
#define KREG_R15        KDIREG_R15
#define KREG_DS         KDIREG_DS
#define KREG_ES         KDIREG_ES
#define KREG_FS         KDIREG_FS
#define KREG_GS         KDIREG_GS
#define KREG_GSBASE     KDIREG_GSBASE
#define KREG_KGSBASE    KDIREG_KGSBASE
#define KREG_TRAPNO     KDIREG_TRAPNO
#define KREG_ERR        KDIREG_ERR
#define KREG_CR2        KDIREG_CR2
#define KREG_CR3        KDIREG_CR3
#define KREG_RIP        KDIREG_RIP
#define KREG_CS         KDIREG_CS
#define KREG_RFLAGS     KDIREG_RFLAGS
#define KREG_RSP        KDIREG_RSP
#define KREG_SS         KDIREG_SS

#define KREG_PC         KREG_RIP
#define KREG_SP         KREG_RSP
#define KREG_FP         KREG_RBP

#else   /* __amd64 */

#define KREG_SAVFP      KDIREG_SAVFP
#define KREG_SAVPC      KDIREG_SAVPC
#define KREG_SS         KDIREG_SS
#define KREG_GS         KDIREG_GS
#define KREG_FS         KDIREG_FS
#define KREG_ES         KDIREG_ES
#define KREG_DS         KDIREG_DS
#define KREG_EDI        KDIREG_EDI
#define KREG_ESI        KDIREG_ESI
#define KREG_EBP        KDIREG_EBP
#define KREG_ESP        KDIREG_ESP
#define KREG_EBX        KDIREG_EBX
#define KREG_EDX        KDIREG_EDX
#define KREG_ECX        KDIREG_ECX
#define KREG_EAX        KDIREG_EAX
#define KREG_TRAPNO     KDIREG_TRAPNO
#define KREG_ERR        KDIREG_ERR
#define KREG_EIP        KDIREG_EIP
#define KREG_CS         KDIREG_CS
#define KREG_EFLAGS     KDIREG_EFLAGS
#define KREG_UESP       KDIREG_UESP

#define KREG_PC         KREG_EIP
#define KREG_SP         KREG_ESP
#define KREG_FP         KREG_EBP
```

```
 122 #endif  /* __amd64 */

 124 #define KREG_EFLAGS_ID_MASK     0x00200000
 125 #define KREG_EFLAGS_ID_SHIFT    21

 127 #define KREG_EFLAGS_VIP_MASK    0x00100000
 128 #define KREG_EFLAGS_VIP_SHIFT   20

 130 #define KREG_EFLAGS_VIF_MASK    0x00080000
 131 #define KREG_EFLAGS_VIF_SHIFT   19

 133 #define KREG_EFLAGS_AC_MASK     0x00040000
 134 #define KREG_EFLAGS_AC_SHIFT    18

 136 #define KREG_EFLAGS_VM_MASK     0x00020000
 137 #define KREG_EFLAGS_VM_SHIFT    17

 139 #define KREG_EFLAGS_RF_MASK     0x00010000
 140 #define KREG_EFLAGS_RF_SHIFT    16

 142 #define KREG_EFLAGS_NT_MASK     0x00004000
 143 #define KREG_EFLAGS_NT_SHIFT    14

 145 #define KREG_EFLAGS_IOPL_MASK   0x00003000
 146 #define KREG_EFLAGS_IOPL_SHIFT  12

 148 #define KREG_EFLAGS_OF_MASK     0x00000800
 149 #define KREG_EFLAGS_OF_SHIFT    11

 151 #define KREG_EFLAGS_DF_MASK     0x00000400
 152 #define KREG_EFLAGS_DF_SHIFT    10

 154 #define KREG_EFLAGS_IF_MASK     0x00000200
 155 #define KREG_EFLAGS_IF_SHIFT    9

 157 #define KREG_EFLAGS_TF_MASK     0x00000100
 158 #define KREG_EFLAGS_TF_SHIFT    8

 160 #define KREG_EFLAGS_SF_MASK     0x00000080
 161 #define KREG_EFLAGS_SF_SHIFT    7

 163 #define KREG_EFLAGS_ZF_MASK     0x00000040
 164 #define KREG_EFLAGS_ZF_SHIFT    6

 166 #define KREG_EFLAGS_AF_MASK     0x00000010
 167 #define KREG_EFLAGS_AF_SHIFT    4

 169 #define KREG_EFLAGS_PF_MASK     0x00000004
 170 #define KREG_EFLAGS_PF_SHIFT    2

 172 #define KREG_EFLAGS_CF_MASK     0x00000001
 173 #define KREG_EFLAGS_CF_SHIFT    0

 175 /* %dr7 */
 176 #define KREG_DRCTL_WP_BASESHIFT 16
 177 #define KREG_DRCTL_WP_INCRSHIFT 4
 178 #define KREG_DRCTL_WP_LENSHIFT  2
 179 #define KREG_DRCTL_WP_LENRWMASK 0xf

 181 #define KREG_DRCTL_WP_EXEC      0
 182 #define KREG_DRCTL_WP_WONLY     1
 183 #define KREG_DRCTL_WP_IORW      2
 184 #define KREG_DRCTL_WP_RW        3

 186 #define KREG_DRCTL_WP_SHIFT(n) \
 187         (KREG_DRCTL_WP_BASESHIFT + KREG_DRCTL_WP_INCRSHIFT * (n))
```



```
 188 #define KREG_DRCTL_WP_MASK(n) \
 189         (KREG_DRCTL_WP_LENRWMASK << KREG_DRCTL_WP_SHIFT(n))
 190 #define KREG_DRCTL_WP_LENRW(n, len, rw) \
 191         ((((len) << KREG_DRCTL_WP_LENSHIFT) | (rw)) << KREG_DRCTL_WP_SHIFT(n))

 193 #define KREG_DRCTL_WPEN_INCRSHIFT 2
 194 #define KREG_DRCTL_WPEN_MASK(n) \
 195         (3 << (KREG_DRCTL_WPEN_INCRSHIFT * (n)))
 196 #define KREG_DRCTL_WPEN(n)      KREG_DRCTL_WPEN_MASK(n)

 198 /* %dr6 */
 199 #define KREG_DRSTAT_BT_MASK     0x00008000
 200 #define KREG_DRSTAT_BS_MASK     0x00004000
 201 #define KREG_DRSTAT_BD_MASK     0x00002000

 203 #define KREG_DRSTAT_WP_MASK(n)  (1 << (n))

 205 #ifdef  __cplusplus
 206 }
______unchanged_portion_omitted_
```

```
**********************************************************
   13668 Fri Apr  6 17:24:58 2018
new/usr/src/cmd/mdb/sparc/kmdb/kvm_isadep.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (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 2006 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

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

/*
 * isa-dependent portions of the kmdb target
 */

#include <mdb/mdb_kreg_impl.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_v9util.h>
#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_umem.h>
#include <kmdb/kmdb_kdi.h>
#include <kmdb/kmdb_dpi.h>
#include <kmdb/kmdb_promif.h>
#include <kmdb/kmdb_asmutil.h>
#include <kmdb/kvm.h>
#include <mdb/mdb.h>

#include <sys/types.h>
#include <sys/stack.h>
#include <sys/regset.h>
#include <sys/sysmacros.h>
#include <sys/bitmap.h>
#include <sys/machtrap.h>
#include <sys/trap.h>

/* Higher than the highest trap number for which we have a specific specifier */
```



```
#define KMT_MAXTRAPNO   0x1ff

#define OP(x)           ((x) >> 30)
#define OP3(x)          (((x) >> 19) & 0x3f)
#define RD(x)           (((x) >> 25) & 0x1f)
#define RS1(x)          (((x) >> 14) & 0x1f)
#define RS2(x)          ((x) & 0x1f)

#define OP_ARITH        0x2

#define OP3_OR          0x02
#define OP3_SAVE        0x3c
#define OP3_RESTORE     0x3d

static int
kmt_stack_iter(mdb_tgt_t *t, const mdb_tgt_gregset_t *gsp,
    mdb_tgt_stack_f *func, void *arg, int cpuid)
{
        const mdb_tgt_gregset_t *grp;
        mdb_tgt_gregset_t gregs;
        kreg_t *kregs = &gregs.kregs[0];
        long nwin, stopwin, canrestore, wp, i, sp;
        long argv[6];

        /*
         * If gsp isn't null, we were asked to dump a trace from a
         * specific location.  The normal iterator can handle that.
         */
        if (gsp != NULL) {
                if (cpuid != DPI_MASTER_CPUID)
                        warn("register set provided - ignoring cpu argument\n");
                return (mdb_kvm_v9stack_iter(t, gsp, func, arg));
        }

        if (kmdb_dpi_get_cpu_state(cpuid) < 0) {
                warn("failed to iterate through stack for cpu %u", cpuid);
                return (DCMD_ERR);
        }

        /*
         * We're being asked to dump the trace for the current CPU.
         * To do that, we need to iterate first through the saved
         * register windors.  If there's more to the trace than that,
         * we'll hand off to the normal iterator.
         */
        if ((grp = kmdb_dpi_get_gregs(cpuid)) == NULL) {
                warn("failed to retrieve registers for cpu %d", cpuid);
                return (DCMD_ERR);
        }

        bcopy(grp, &gregs, sizeof (mdb_tgt_gregset_t));

        wp = kregs[KREG_CWP];
        canrestore = kregs[KREG_CANRESTORE];
        nwin = kmdb_dpi_get_nwin(cpuid);
        stopwin = ((wp + nwin) - canrestore - 1) % nwin;

        mdb_dprintf(MDB_DBG_KMOD, "dumping cwp = %lu, canrestore = %lu, "
            "stopwin = %lu\n", wp, canrestore, stopwin);

        for (;;) {
                struct rwindow rwin;

                for (i = 0; i < 6; i++)
                        argv[i] = kregs[KREG_I0 + i];
```

```
122                 if (kregs[KREG_PC] != 0 &&
123                     func(arg, kregs[KREG_PC], 6, argv, &gregs) != 0)
124                         return (0);

126                 kregs[KREG_PC] = kregs[KREG_I7];
127                 kregs[KREG_NPC] = kregs[KREG_PC] + 4;

129                 if ((sp = kregs[KREG_FP] + STACK_BIAS) == STACK_BIAS || sp == 0)
130                         return (0); /* Stop if we're at the end of stack */

132                 if (sp & (STACK_ALIGN - 1))
133                         return (set_errno(EMDB_STKALIGN));

135                 wp = (wp + nwin - 1) % nwin;

137                 if (wp == stopwin)
138                         break;

140                 bcopy(&kregs[KREG_I0], &kregs[KREG_O0], 8 * sizeof (kreg_t));

142                 if (kmdb_dpi_get_rwin(cpuid, wp, &rwin) < 0) {
143                         warn("unable to get registers from window %ld\n", wp);
144                         return (-1);
145                 }

147                 for (i = 0; i < 8; i++)
148                         kregs[KREG_L0 + i] = (uintptr_t)rwin.rw_local[i];
149                 for (i = 0; i < 8; i++)
150                         kregs[KREG_I0 + i] = (uintptr_t)rwin.rw_in[i];
151         }

153         mdb_dprintf(MDB_DBG_KMOD, "dumping wp %ld and beyond normally\n", wp);

155         /*
156          * hack - if we null out pc here, iterator won't print the frame
157          * that corresponds to the current set of registers.  That's what we
158          * want because we just printed them above.
159          */
160         kregs[KREG_PC] = 0;
161         return (mdb_kvm_v9stack_iter(t, &gregs, func, arg));
162 }
```

_____unchanged_portion_omitted_

```
380 /*ARGSUSED*/
381 int
382 kmt_step_branch(mdb_tgt_t *t)
383 {
384         return (set_errno(EMDB_TGTHWNOTSUP));
385 }
```

```
380 static const char *
381 regno2name(int idx)
382 {
383         const mdb_tgt_regdesc_t *rd;

385         for (rd = mdb_sparcv9_kregs; rd->rd_name != NULL; rd++) {
386                 if (idx == rd->rd_num)
387                         return (rd->rd_name);
388         }

390         ASSERT(rd->rd_name != NULL);

392         return ("unknown");
393 }
```

_____unchanged_portion_omitted_

```
**********************************************************
   15729 Fri Apr  6 17:24:59 2018
new/usr/src/cmd/mdb/sparc/mdb/kvm_v7dep.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

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

/*
 * Libkvm Kernel Target SPARC v7 component
 *
 * This file provides the ISA-dependent portion of the libkvm kernel target.
 * For more details on the implementation refer to mdb_kvm.c.
 */

#include <sys/types.h>
#include <sys/machtypes.h>
#include <sys/regset.h>
#include <sys/frame.h>
#include <sys/stack.h>
#include <sys/sysmacros.h>
#include <sys/panic.h>
#include <strings.h>

#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_disasm.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_conf.h>
#include <mdb/mdb_kreg.h>
#include <mdb/mdb_kvm.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb.h>

/*
 * The mdb_tgt_gregset type is opaque to callers of the target interface
```



```
 * and to our own target common code.  We now can define it explicitly.
 */
struct mdb_tgt_gregset {
        kreg_t kregs[KREG_NGREG];
};
```

*unchanged_portion_omitted_*

```
const mdb_tgt_ops_t kt_sparcv7_ops = {
        kt_setflags,                            /* t_setflags */
        kt_setcontext,                          /* t_setcontext */
        kt_activate,                            /* t_activate */
        kt_deactivate,                          /* t_deactivate */
        (void (*)()) mdb_tgt_nop,               /* t_periodic */
        kt_destroy,                             /* t_destroy */
        kt_name,                                /* t_name */
        (const char *(*)()) mdb_conf_isa,       /* t_isa */
        kt_platform,                            /* t_platform */
        kt_uname,                               /* t_uname */
        kt_dmodel,                              /* t_dmodel */
        kt_aread,                               /* t_aread */
        kt_awrite,                              /* t_awrite */
        kt_vread,                               /* t_vread */
        kt_vwrite,                              /* t_vwrite */
        kt_pread,                               /* t_pread */
        kt_pwrite,                              /* t_pwrite */
        kt_fread,                               /* t_fread */
        kt_fwrite,                              /* t_fwrite */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_ioread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_iowrite */
        kt_vtop,                                /* t_vtop */
        kt_lookup_by_name,                      /* t_lookup_by_name */
        kt_lookup_by_addr,                      /* t_lookup_by_addr */
        kt_symbol_iter,                         /* t_symbol_iter */
        kt_mapping_iter,                        /* t_mapping_iter */
        kt_object_iter,                         /* t_object_iter */
        kt_addr_to_map,                         /* t_addr_to_map */
        kt_name_to_map,                         /* t_name_to_map */
        kt_addr_to_ctf,                         /* t_addr_to_ctf */
        kt_name_to_ctf,                         /* t_name_to_ctf */
        kt_status,                              /* t_status */
        (int (*)()) mdb_tgt_notsup,             /* t_run */
        (int (*)()) mdb_tgt_notsup,             /* t_step */
        (int (*)()) mdb_tgt_notsup,             /* t_step_out */
        (int (*)()) mdb_tgt_notsup,             /* t_step_branch */
        (int (*)()) mdb_tgt_notsup,             /* t_next */
        (int (*)()) mdb_tgt_notsup,             /* t_cont */
        (int (*)()) mdb_tgt_notsup,             /* t_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_vbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_sbrkpt */
        (int (*)()) mdb_tgt_null,               /* t_add_pwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_vwapt */
        (int (*)()) mdb_tgt_null,               /* t_add_iowapt */
        (int (*)()) mdb_tgt_null,               /* t_add_sysenter */
        (int (*)()) mdb_tgt_null,               /* t_add_sysexit */
        (int (*)()) mdb_tgt_null,               /* t_add_signal */
        (int (*)()) mdb_tgt_null,               /* t_add_fault */
        kt_getareg,                             /* t_getareg */
        kt_putareg,                             /* t_putareg */
        kt_stack_iter,                          /* t_stack_iter */
        (int (*)()) mdb_tgt_notsup              /* t_auxv */
};
```

*unchanged_portion_omitted_*

```
**********************************************************
   10883 Fri Apr  6 17:24:59 2018
new/usr/src/cmd/mdb/sparc/mdb/kvm_v9dep.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

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

/*
 * Libkvm Kernel Target SPARC v9 component
 *
 * This file provides the ISA-dependent portion of the libkvm kernel target.
 * For more details on the implementation refer to mdb_kvm.c.  The SPARC v9
 * ISA code is actually compiled into *both* the sparcv7 and sparcv9 MDB
 * binaries because we need to deal with the sparcv9 CPU registers when
 * debugging a 32-bit crash dump from a kernel running on a sparcv9 CPU.
 */

#ifndef __sparcv9cpu
#define __sparcv9cpu
#endif

#include <sys/types.h>
#include <sys/machtypes.h>
#include <sys/regset.h>
#include <sys/frame.h>
#include <sys/stack.h>
#include <sys/sysmacros.h>
#include <sys/panic.h>
#include <strings.h>

#include <mdb/mdb_target_impl.h>
#include <mdb/mdb_disasm.h>
#include <mdb/mdb_modapi.h>
#include <mdb/mdb_conf.h>
#include <mdb/mdb_kreg_impl.h>
```



```
#include <mdb/mdb_v9util.h>
#include <mdb/mdb_kvm.h>
#include <mdb/mdb_err.h>
#include <mdb/mdb_debug.h>
#include <mdb/mdb.h>

#ifndef STACK_BIAS
#define STACK_BIAS      0
#endif

static int
kt_getareg(mdb_tgt_t *t, mdb_tgt_tid_t tid,
    const char *rname, mdb_tgt_reg_t *rp)
{
        const mdb_tgt_regdesc_t *rdp;
        kt_data_t *kt = t->t_data;

        if (tid != kt->k_tid)
                return (set_errno(EMDB_NOREGS));

        for (rdp = kt->k_rds; rdp->rd_name != NULL; rdp++) {
                if (strcmp(rname, rdp->rd_name) == 0) {
                        *rp = kt->k_regs->kregs[rdp->rd_num];
                        return (0);
                }
        }

        return (set_errno(EMDB_BADREG));
}
______unchanged_portion_omitted_

const mdb_tgt_ops_t kt_sparcv9_ops = {
        kt_setflags,                            /* t_setflags */
        kt_setcontext,                          /* t_setcontext */
        kt_activate,                            /* t_activate */
        kt_deactivate,                          /* t_deactivate */
        (void (*)()) mdb_tgt_nop,               /* t_periodic */
        kt_destroy,                             /* t_destroy */
        kt_name,                                /* t_name */
        (const char *(*)()) mdb_conf_isa,       /* t_isa */
        kt_platform,                            /* t_platform */
        kt_uname,                               /* t_uname */
        kt_dmodel,                              /* t_dmodel */
        kt_aread,                               /* t_aread */
        kt_awrite,                              /* t_awrite */
        kt_vread,                               /* t_vread */
        kt_vwrite,                              /* t_vwrite */
        kt_pread,                               /* t_pread */
        kt_pwrite,                              /* t_pwrite */
        kt_fread,                               /* t_fread */
        kt_fwrite,                              /* t_fwrite */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_ioread */
        (ssize_t (*)()) mdb_tgt_notsup,         /* t_iowrite */
        kt_vtop,                                /* t_vtop */
        kt_lookup_by_name,                      /* t_lookup_by_name */
        kt_lookup_by_addr,                      /* t_lookup_by_addr */
        kt_symbol_iter,                         /* t_symbol_iter */
        kt_mapping_iter,                        /* t_mapping_iter */
        kt_object_iter,                         /* t_object_iter */
        kt_addr_to_map,                         /* t_addr_to_map */
        kt_name_to_map,                         /* t_name_to_map */
        kt_addr_to_ctf,                         /* t_addr_to_ctf */
        kt_name_to_ctf,                         /* t_name_to_ctf */
        kt_status,                              /* t_status */
        (int (*)()) mdb_tgt_notsup,             /* t_run */
        (int (*)()) mdb_tgt_notsup,             /* t_step */
```

```
219         (int (*)()) mdb_tgt_notsup,             /* t_step_out */
220         (int (*)()) mdb_tgt_notsup,             /* t_step_branch */
220         (int (*)()) mdb_tgt_notsup,             /* t_next */
221         (int (*)()) mdb_tgt_notsup,             /* t_cont */
222         (int (*)()) mdb_tgt_notsup,             /* t_signal */
223         (int (*)()) mdb_tgt_null,               /* t_add_vbrkpt */
224         (int (*)()) mdb_tgt_null,               /* t_add_sbrkpt */
225         (int (*)()) mdb_tgt_null,               /* t_add_pwapt */
226         (int (*)()) mdb_tgt_null,               /* t_add_iowapt */
227         (int (*)()) mdb_tgt_null,               /* t_add_vwapt */
228         (int (*)()) mdb_tgt_null,               /* t_add_sysenter */
229         (int (*)()) mdb_tgt_null,               /* t_add_sysexit */
230         (int (*)()) mdb_tgt_null,               /* t_add_signal */
231         (int (*)()) mdb_tgt_null,               /* t_add_fault */
232         kt_getareg,                             /* t_getareg */
233         kt_putareg,                             /* t_putareg */
234         mdb_kvm_v9stack_iter,                   /* t_stack_iter */
235         (int (*)()) mdb_tgt_notsup              /* t_auxv */
236 };
```

*unchanged_portion_omitted_*

```
**********************************************************
   12018 Fri Apr  6 17:24:59 2018
new/usr/src/uts/common/sys/sysmacros.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*        All Rights Reserved   */


/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
 *
 * Copyright 2018 Joyent Inc.
 */

#ifndef _SYS_SYSMACROS_H
#define _SYS_SYSMACROS_H

#include <sys/param.h>
#include <sys/stddef.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Some macros for units conversion
 */
/*
 * Disk blocks (sectors) and bytes.
 */
#define dtob(DD)        ((DD) << DEV_BSHIFT)
#define btod(BB)        (((BB) + DEV_BSIZE - 1) >> DEV_BSHIFT)
#define btodt(BB)       ((BB) >> DEV_BSHIFT)
#define lbtod(BB)       (((offset_t)(BB) + DEV_BSIZE - 1) >> DEV_BSHIFT)

/* common macros */
#ifndef MIN
#define MIN(a, b)       ((a) < (b) ? (a) : (b))
#endif
#ifndef MAX
```



```
#define MAX(a, b)       ((a) < (b) ? (b) : (a))
#endif
#ifndef ABS
#define ABS(a)          ((a) < 0 ? -(a) : (a))
#endif
#ifndef SIGNOF
#define SIGNOF(a)       ((a) < 0 ? -1 : (a) > 0)
#endif

#ifdef _KERNEL

/*
 * Convert a single byte to/from binary-coded decimal (BCD).
 */
extern unsigned char byte_to_bcd[256];
extern unsigned char bcd_to_byte[256];

#define BYTE_TO_BCD(x)  byte_to_bcd[(x) & 0xff]
#define BCD_TO_BYTE(x)  bcd_to_byte[(x) & 0xff]

#endif  /* _KERNEL */

/*
 * WARNING: The device number macros defined here should not be used by device
 * drivers or user software. Device drivers should use the device functions
 * defined in the DDI/DKI interface (see also ddi.h). Application software
 * should make use of the library routines available in makedev(3). A set of
 * new device macros are provided to operate on the expanded device number
 * format supported in SVR4. Macro versions of the DDI device functions are
 * provided for use by kernel proper routines only. Macro routines bmajor(),
 * major(), minor(), emajor(), eminor(), and makedev() will be removed or
 * their definitions changed at the next major release following SVR4.
 */

#define O_BITSMAJOR     7       /* # of SVR3 major device bits */
#define O_BITSMINOR     8       /* # of SVR3 minor device bits */
#define O_MAXMAJ        0x7f    /* SVR3 max major value */
#define O_MAXMIN        0xff    /* SVR3 max minor value */


#define L_BITSMAJOR32   14      /* # of SVR4 major device bits */
#define L_BITSMINOR32   18      /* # of SVR4 minor device bits */
#define L_MAXMAJ32      0x3fff  /* SVR4 max major value */
#define L_MAXMIN32      0x3ffff /* MAX minor for 3b2 software drivers. */
                                /* For 3b2 hardware devices the minor is */
                                /* restricted to 256 (0-255) */

#ifdef _LP64
#define L_BITSMAJOR     32      /* # of major device bits in 64-bit Solaris */
#define L_BITSMINOR     32      /* # of minor device bits in 64-bit Solaris */
#define L_MAXMAJ        0xffffffffu    /* max major value */
#define L_MAXMIN        0xffffffffu    /* max minor value */
#else
#define L_BITSMAJOR     L_BITSMAJOR32
#define L_BITSMINOR     L_BITSMINOR32
#define L_MAXMAJ        L_MAXMAJ32
#define L_MAXMIN        L_MAXMIN32
#endif

#ifdef _KERNEL

/* major part of a device internal to the kernel */

#define major(x)        (major_t)((((unsigned)(x)) >> O_BITSMINOR) & O_MAXMAJ)
#define bmajor(x)       (major_t)((((unsigned)(x)) >> O_BITSMINOR) & O_MAXMAJ)
```

```
126 /* get internal major part of expanded device number */

128 #define getmajor(x)     (major_t)((((dev_t)(x)) >> L_BITSMINOR) & L_MAXMAJ)

130 /* minor part of a device internal to the kernel */

132 #define minor(x)        (minor_t)((x) & O_MAXMIN)

134 /* get internal minor part of expanded device number */

136 #define getminor(x)     (minor_t)((x) & L_MAXMIN)

138 #else   /* _KERNEL */

140 /* major part of a device external from the kernel (same as emajor below) */

142 #define major(x)        (major_t)((((unsigned)(x)) >> O_BITSMINOR) & O_MAXMAJ)

144 /* minor part of a device external from the kernel  (same as eminor below) */

146 #define minor(x)        (minor_t)((x) & O_MAXMIN)

148 #endif  /* _KERNEL */

150 /* create old device number */

152 #define makedev(x, y) (unsigned short)(((x) << O_BITSMINOR) | ((y) & O_MAXMIN))

154 /* make an new device number */

156 #define makedevice(x, y) (dev_t)(((dev_t)(x) << L_BITSMINOR) | ((y) & L_MAXMIN))


159 /*
160  * emajor() allows kernel/driver code to print external major numbers
161  * eminor() allows kernel/driver code to print external minor numbers
162  */

164 #define emajor(x) \
165         (major_t)(((unsigned int)(x) >> O_BITSMINOR) > O_MAXMAJ) ? \
166             NODEV : (((unsigned int)(x) >> O_BITSMINOR) & O_MAXMAJ)

168 #define eminor(x) \
169         (minor_t)((x) & O_MAXMIN)

171 /*
172  * get external major and minor device
173  * components from expanded device number
174  */
175 #define getemajor(x)    (major_t)((((dev_t)(x) >> L_BITSMINOR) > L_MAXMAJ) ? \
176                             NODEV : (((dev_t)(x) >> L_BITSMINOR) & L_MAXMAJ))
177 #define geteminor(x)    (minor_t)((x) & L_MAXMIN)

179 /*
180  * These are versions of the kernel routines for compressing and
181  * expanding long device numbers that don't return errors.
182  */
183 #if (L_BITSMAJOR32 == L_BITSMAJOR) && (L_BITSMINOR32 == L_BITSMINOR)

185 #define DEVCMPL(x)      (x)
186 #define DEVEXPL(x)      (x)

188 #else

190 #define DEVCMPL(x)      \
191         (dev32_t)((((x) >> L_BITSMINOR) > L_MAXMAJ32 || \
```



```
192             ((x) & L_MAXMIN) > L_MAXMIN32) ? NODEV32 : \
193             ((((x) >> L_BITSMINOR) << L_BITSMINOR32) | ((x) & L_MAXMIN32)))

195 #define DEVEXPL(x)      \
196         (((x) == NODEV32) ? NODEV : \
197         makedevice(((x) >> L_BITSMINOR32) & L_MAXMAJ32, (x) & L_MAXMIN32))

199 #endif /* L_BITSMAJOR32 ... */

201 /* convert to old (SVR3.2) dev format */

203 #define cmpdev(x) \
204         (o_dev_t)((((x) >> L_BITSMINOR) > O_MAXMAJ || \
205             ((x) & L_MAXMIN) > O_MAXMIN) ? NODEV : \
206             ((((x) >> L_BITSMINOR) << O_BITSMINOR) | ((x) & O_MAXMIN)))

208 /* convert to new (SVR4) dev format */

210 #define expdev(x) \
211         (dev_t)(((dev_t)(((x) >> O_BITSMINOR) & O_MAXMAJ) << L_BITSMINOR) | \
212             ((x) & O_MAXMIN))

214 /*
215  * Macro for checking power of 2 address alignment.
216  */
217 #define IS_P2ALIGNED(v, a) ((((uintptr_t)(v)) & ((uintptr_t)(a) - 1)) == 0)

219 /*
220  * Macros for counting and rounding.
221  */
222 #define howmany(x, y)   (((x)+((y)-1))/(y))
223 #define roundup(x, y)   ((((x)+((y)-1))/(y))*(y))

225 /*
226  * Macro to determine if value is a power of 2
227  */
228 #define ISP2(x)         (((x) & ((x) - 1)) == 0)

230 /*
231  * Macros for various sorts of alignment and rounding.  The "align" must
232  * be a power of 2.  Often times it is a block, sector, or page.
233  */

235 /*
236  * return x rounded down to an align boundary
237  * eg, P2ALIGN(1200, 1024) == 1024 (1*align)
238  * eg, P2ALIGN(1024, 1024) == 1024 (1*align)
239  * eg, P2ALIGN(0x1234, 0x100) == 0x1200 (0x12*align)
240  * eg, P2ALIGN(0x5600, 0x100) == 0x5600 (0x56*align)
241  */
242 #define P2ALIGN(x, align)               ((x) & -(align))

244 /*
245  * return x % (mod) align
246  * eg, P2PHASE(0x1234, 0x100) == 0x34 (x-0x12*align)
247  * eg, P2PHASE(0x5600, 0x100) == 0x00 (x-0x56*align)
248  */
249 #define P2PHASE(x, align)               ((x) & ((align) - 1))

251 /*
252  * return how much space is left in this block (but if it's perfectly
253  * aligned, return 0).
254  * eg, P2NPHASE(0x1234, 0x100) == 0xcc (0x13*align-x)
255  * eg, P2NPHASE(0x5600, 0x100) == 0x00 (0x56*align-x)
256  */
257 #define P2NPHASE(x, align)              (-(x) & ((align) - 1))
```

```
 259 /*
 260  * return x rounded up to an align boundary
 261  * eg, P2ROUNDUP(0x1234, 0x100) == 0x1300 (0x13*align)
 262  * eg, P2ROUNDUP(0x5600, 0x100) == 0x5600 (0x56*align)
 263  */
 264 #define P2ROUNDUP(x, align)             (-(-(x) & -(align)))

 266 /*
 267  * return the ending address of the block that x is in
 268  * eg, P2END(0x1234, 0x100) == 0x12ff (0x13*align - 1)
 269  * eg, P2END(0x5600, 0x100) == 0x56ff (0x57*align - 1)
 270  */
 271 #define P2END(x, align)                 (-(~(x) & -(align)))

 273 /*
 274  * return x rounded up to the next phase (offset) within align.
 275  * phase should be < align.
 276  * eg, P2PHASEUP(0x1234, 0x100, 0x10) == 0x1310 (0x13*align + phase)
 277  * eg, P2PHASEUP(0x5600, 0x100, 0x10) == 0x5610 (0x56*align + phase)
 278  */
 279 #define P2PHASEUP(x, align, phase)      ((phase) - (((phase) - (x)) & -(align)))

 281 /*
 282  * return TRUE if adding len to off would cause it to cross an align
 283  * boundary.
 284  * eg, P2BOUNDARY(0x1234, 0xe0, 0x100) == TRUE (0x1234 + 0xe0 == 0x1314)
 285  * eg, P2BOUNDARY(0x1234, 0x50, 0x100) == FALSE (0x1234 + 0x50 == 0x1284)
 286  */
 287 #define P2BOUNDARY(off, len, align) \
 288         (((off) ^ ((off) + (len) - 1)) > (align) - 1)

 290 /*
 291  * Return TRUE if they have the same highest bit set.
 292  * eg, P2SAMEHIGHBIT(0x1234, 0x1001) == TRUE (the high bit is 0x1000)
 293  * eg, P2SAMEHIGHBIT(0x1234, 0x3010) == FALSE (high bit of 0x3010 is 0x2000)
 294  */
 295 #define P2SAMEHIGHBIT(x, y)             (((x) ^ (y)) < ((x) & (y)))

 297 /*
 298  * Typed version of the P2* macros.  These macros should be used to ensure
 299  * that the result is correctly calculated based on the data type of (x),
 300  * which is passed in as the last argument, regardless of the data
 301  * type of the alignment.  For example, if (x) is of type uint64_t,
 302  * and we want to round it up to a page boundary using "PAGESIZE" as
 303  * the alignment, we can do either
 304  *      P2ROUNDUP(x, (uint64_t)PAGESIZE)
 305  * or
 306  *      P2ROUNDUP_TYPED(x, PAGESIZE, uint64_t)
 307  */
 308 #define P2ALIGN_TYPED(x, align, type)   \
 309         ((type)(x) & -(type)(align))
 310 #define P2PHASE_TYPED(x, align, type)   \
 311         ((type)(x) & ((type)(align) - 1))
 312 #define P2NPHASE_TYPED(x, align, type)  \
 313         (-(type)(x) & ((type)(align) - 1))
 314 #define P2ROUNDUP_TYPED(x, align, type) \
 315         (-(-(type)(x) & -(type)(align)))
 316 #define P2END_TYPED(x, align, type)     \
 317         (-(~(type)(x) & -(type)(align)))
 318 #define P2PHASEUP_TYPED(x, align, phase, type)  \
 319         ((type)(phase) - (((type)(phase) - (type)(x)) & -(type)(align)))
 320 #define P2CROSS_TYPED(x, y, align, type)        \
 321         (((type)(x) ^ (type)(y)) > (type)(align) - 1)
 322 #define P2SAMEHIGHBIT_TYPED(x, y, type) \
 323         (((type)(x) ^ (type)(y)) < ((type)(x) & (type)(y)))
```



```
 325 /*
 326  * Macros to atomically increment/decrement a variable.  mutex and var
 327  * must be pointers.
 328  */
 329 #define INCR_COUNT(var, mutex) mutex_enter(mutex), (*(var))++, mutex_exit(mutex)
 330 #define DECR_COUNT(var, mutex) mutex_enter(mutex), (*(var))--, mutex_exit(mutex)

 332 /*
 333  * Macros to declare bitfields - the order in the parameter list is
 334  * Low to High - that is, declare bit 0 first.  We only support 8-bit bitfields
 335  * because if a field crosses a byte boundary it's not likely to be meaningful
 336  * without reassembly in its nonnative endianness.
 337  */
 338 #if defined(_BIT_FIELDS_LTOH)
 339 #define DECL_BITFIELD2(_a, _b)                          \
 340         uint8_t _a, _b
 341 #define DECL_BITFIELD3(_a, _b, _c)                      \
 342         uint8_t _a, _b, _c
 343 #define DECL_BITFIELD4(_a, _b, _c, _d)                  \
 344         uint8_t _a, _b, _c, _d
 345 #define DECL_BITFIELD5(_a, _b, _c, _d, _e)              \
 346         uint8_t _a, _b, _c, _d, _e
 347 #define DECL_BITFIELD6(_a, _b, _c, _d, _e, _f)          \
 348         uint8_t _a, _b, _c, _d, _e, _f
 349 #define DECL_BITFIELD7(_a, _b, _c, _d, _e, _f, _g)      \
 350         uint8_t _a, _b, _c, _d, _e, _f, _g
 351 #define DECL_BITFIELD8(_a, _b, _c, _d, _e, _f, _g, _h)  \
 352         uint8_t _a, _b, _c, _d, _e, _f, _g, _h
 353 #elif defined(_BIT_FIELDS_HTOL)
 354 #define DECL_BITFIELD2(_a, _b)                          \
 355         uint8_t _b, _a
 356 #define DECL_BITFIELD3(_a, _b, _c)                      \
 357         uint8_t _c, _b, _a
 358 #define DECL_BITFIELD4(_a, _b, _c, _d)                  \
 359         uint8_t _d, _c, _b, _a
 360 #define DECL_BITFIELD5(_a, _b, _c, _d, _e)              \
 361         uint8_t _e, _d, _c, _b, _a
 362 #define DECL_BITFIELD6(_a, _b, _c, _d, _e, _f)          \
 363         uint8_t _f, _e, _d, _c, _b, _a
 364 #define DECL_BITFIELD7(_a, _b, _c, _d, _e, _f, _g)      \
 365         uint8_t _g, _f, _e, _d, _c, _b, _a
 366 #define DECL_BITFIELD8(_a, _b, _c, _d, _e, _f, _g, _h)  \
 367         uint8_t _h, _g, _f, _e, _d, _c, _b, _a
 368 #else
 369 #error  One of _BIT_FIELDS_LTOH or _BIT_FIELDS_HTOL must be defined
 370 #endif  /* _BIT_FIELDS_LTOH */

 372 #if !defined(ARRAY_SIZE)
 370 /* avoid any possibility of clashing with <stddef.h> version */
 371 #if (defined(_KERNEL) || defined(_FAKE_KERNEL)) && !defined(_KMEMUSER)

 373 #define ARRAY_SIZE(x)   (sizeof (x) / sizeof (x[0]))
 374 #endif

 375 #endif /* _KERNEL, !_KMEMUSER */

 376 #ifdef  __cplusplus
 377 }
_____unchanged_portion_omitted_
```

```
**********************************************************
    6325 Fri Apr  6 17:24:59 2018
new/usr/src/uts/i86pc/Makefile.files
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
#
# Copyright (c) 2010, Intel Corporation.
# Copyright 2018 Joyent, Inc.
# Copyright 2017 Joyent, Inc.
#
#       This Makefile defines file modules in the directory uts/i86pc
#       and its children. These are the source files which are i86pc
#       "implementation architecture" dependent.
#

#
#       object lists
#
CORE_OBJS +=                    \
        acpi_stubs.o            \
        biosdisk.o              \
        bios_call.o             \
        cbe.o                   \
        cmi.o                   \
        cmi_hw.o                \
        cms.o                   \
        comm_page.o             \
        confunix.o              \
        cpu_idle.o              \
        cpuid.o                 \
        cpuid_subr.o            \
        cpupm.o                 \
        cpupm_mach.o            \
        cpupm_amd.o             \
        cpupm_intel.o           \
        cpupm_throttle.o        \
        cpu_acpi.o              \
        dis_tables.o            \
        ddi_impl.o              \
        dtrace_subr.o           \
        dvma.o                  \
```



```
        fpu_subr.o              \
        fakebop.o               \
        fastboot.o              \
        fb_swtch.o              \
        graphics.o              \
        hardclk.o               \
        hat_i86.o               \
        hat_kdi.o               \
        hment.o                 \
        hold_page.o             \
        hrtimers.o              \
        htable.o                \
        hypercall.o             \
        hypersubr.o             \
        i86_mmu.o               \
        ibft.o                  \
        instr_size.o            \
        intr.o                  \
        kboot_mmu.o             \
        kdi_subr.o              \
        kdi_idt.o               \
        kdi_idthdl.o            \
        kdi_asm.o               \
        lgrpplat.o              \
        mach_kdi.o              \
        mach_sysconfig.o        \
        machdep.o               \
        mem_config.o            \
        mem_config_stubs.o      \
        mem_config_arch.o       \
        memlist_new.o           \
        memnode.o               \
        microcode.o             \
        microfind.o             \
        mlsetup.o               \
        mp_call.o               \
        mp_implfuncs.o          \
        mp_machdep.o            \
        mp_pc.o                 \
        mp_startup.o            \
        memscrub.o              \
        mpcore.o                \
        notes.o                 \
        pci_bios.o              \
        pci_cfgacc.o            \
        pci_cfgacc_x86.o        \
        pci_cfgspace.o          \
        pci_mech1.o             \
        pci_mech1_amd.o         \
        pci_mech2.o             \
        pci_neptune.o           \
        pci_orion.o             \
        pmem.o                  \
        ppage.o                 \
        pwrnow.o                \
        speedstep.o             \
        ssp.o                   \
        startup.o               \
        timestamp.o             \
        todpc_subr.o            \
        trap.o                  \
        turbo.o                 \
        vm_machdep.o            \
        xpv_platform.o          \
        x_call.o
```

```
124 #
125 #       Add the SMBIOS subsystem object files directly to the list of objects
126 #       built into unix itself; this is all common code except for smb_dev.c.
127 #
128 CORE_OBJS += $(SMBIOS_OBJS)

130 #
131 # These get compiled twice:
132 # - once in the dboot (direct boot) identity mapped code
133 # - once for use during early startup in unix
134 #
135 BOOT_DRIVER_OBJS =              \
136         boot_console.o          \
137         boot_keyboard.o         \
138         boot_keyboard_table.o   \
139         boot_vga.o              \
140         boot_mmu.o              \
141         dboot_multiboot2.o      \
142         $(FONT_OBJS)

144 CORE_OBJS += $(BOOT_DRIVER_OBJS)

146 #
147 #       locore.o is special. It must be the first file relocated so that it
148 #       it is relocated just where its name implies.
149 #
150 SPECIAL_OBJS_32 +=              \
151         locore.o                \
152         fast_trap_asm.o         \
153         interrupt.o             \
154         syscall_asm.o

156 SPECIAL_OBJS_64 +=              \
157         locore.o                \
158         fast_trap_asm.o         \
159         interrupt.o             \
160         syscall_asm_amd64.o     \
161         kpti_trampolines.o
161         syscall_asm_amd64.o

163 SPECIAL_OBJS += $(SPECIAL_OBJS_$(CLASS))

165 #
166 # Objects that get compiled into the identity mapped PT_LOAD section of unix
167 # to handle the earliest part of booting.
168 #
169 DBOOT_OBJS_32 =

171 DBOOT_OBJS_64 += dboot_elfload.o

173 DBOOT_OBJS +=                   \
174         dboot_asm.o             \
175         dboot_grub.o            \
176         dboot_printf.o          \
177         dboot_startkern.o       \
178         memcpy.o                \
179         memset.o                \
180         muldiv.o                \
181         sha1.o                  \
182         string.o                \
183         $(BOOT_DRIVER_OBJS)     \
184         $(DBOOT_OBJS_$(CLASS))

186 #
187 #                       driver and misc modules
188 #
```



```
189 GFX_PRIVATE_OBJS        += gfx_private.o gfxp_pci.o gfxp_segmap.o \
190                            gfxp_devmap.o gfxp_vgatext.o gfxp_vm.o vgasubr.o
191 FIPE_OBJS += fipe_drv.o fipe_pm.o
192 IOAT_OBJS += ioat.o ioat_rs.o ioat_ioctl.o ioat_chan.o
193 ISANEXUS_OBJS += isa.o dma_engine.o i8237A.o
194 PCIE_MISC_OBJS += pcie_acpi.o pciehpc_acpi.o pcie_x86.o
195 PCI_E_NEXUS_OBJS += npe.o npe_misc.o
196 PCI_E_NEXUS_OBJS += pci_common.o pci_kstats.o pci_tools.o
197 PCINEXUS_OBJS += pci.o pci_common.o pci_kstats.o pci_tools.o
198 PCPLUSMP_OBJS += apic.o apic_regops.o psm_common.o apic_introp.o        \
199                          mp_platform_common.o mp_platform_misc.o        \
200                          hpet_acpi.o apic_common.o apic_timer.o
201 APIX_OBJS += apix.o apic_regops.o psm_common.o apix_intr.o apix_utils.o \
202                apix_irm.o mp_platform_common.o hpet_acpi.o apic_common.o \
203                apic_timer.o apix_regops.o


206 ACPI_DRV_OBJS   += acpi_drv.o acpi_video.o
207 ACPINEX_OBJS    += acpinex_drv.o acpinex_event.o

209 CPUDRV_OBJS     += \
210         cpudrv.o \
211         cpudrv_mach.o

213 PPM_OBJS        += ppm_subr.o ppm.o ppm_plat.o

215 ACPIPPM_OBJS    += acpippm.o acpisleep.o
216 ACPIDEV_OBJS += acpidev_drv.o \
217         acpidev_scope.o acpidev_device.o \
218         acpidev_container.o \
219         acpidev_cpu.o \
220         acpidev_dr.o \
221         acpidev_memory.o \
222         acpidev_pci.o \
223         acpidev_resource.o \
224         acpidev_util.o

226 DRMACH_ACPI_OBJS += drmach_acpi.o dr_util.o drmach_err.o

228 DR_OBJS += dr.o dr_cpu.o dr_err.o dr_io.o dr_mem_acpi.o dr_quiesce.o dr_util.o

230 ROOTNEX_OBJS += rootnex.o immu.o immu_dmar.o immu_dvma.o \
231                 immu_intrmap.o immu_qinv.o immu_regs.o

233 TZMON_OBJS      += tzmon.o
234 UPPC_OBJS += uppc.o psm_common.o
235 XSVC_OBJS += xsvc.o
236 AMD_IOMMU_OBJS +=       amd_iommu.o amd_iommu_impl.o amd_iommu_acpi.o \
237                         amd_iommu_cmd.o amd_iommu_log.o amd_iommu_page_tables.o

239 #
240 #       Build up defines and paths.
241 #
242 ALL_DEFS        += -Di86pc
243 INC_PATH        += -I$(UTSBASE)/i86pc -I$(SRC)/common
244 INC_PATH        += -I$(UTSBASE)/i86xpv -I$(UTSBASE)/common/xen

246 #
247 # Since the assym files are derived, the dependencies must be explicit for
248 # all files including this file. (This is only actually required in the
249 # instance when the .nse_depinfo file does not exist.) It may seem that
250 # the lint targets should also have a similar dependency, but they don't
251 # since only C headers are included when #defined(__lint) is true.
252 #

254 ASSYM_DEPS      +=              \
```

```
255         copy.o                  \
256         desctbls_asm.o          \
257         ddi_i86_asm.o           \
258         exception.o             \
259         fast_trap_asm.o         \
260         float.o                 \
261         i86_subr.o              \
262         interrupt.o             \
263         lock_prim.o             \
264         locore.o                \
265         mpcore.o                \
266         sseblk.o                \
267         swtch.o                 \
268         syscall_asm.o           \
269         syscall_asm_amd64.o     \
270         kpti_trampolines.o      \
271         cpr_wakecode.o

273 CPR_IMPL_OBJS   = cpr_impl.o    cpr_wakecode.o

275 $(KDI_ASSYM_DEPS:%=$(OBJS_DIR)/%):      $(DSF_DIR)/$(OBJS_DIR)/kdi_assym.h

276 ASSYM_DEPS += kdi_asm.o
```

```
**********************************************************
    9011 Fri Apr  6 17:24:59 2018
new/usr/src/uts/i86pc/io/gfx_private/gfxp_vm.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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.
 */

#include <sys/debug.h>
#include <sys/types.h>
#include <sys/param.h>
#include <sys/time.h>
#include <sys/buf.h>
#include <sys/errno.h>
#include <sys/systm.h>
#include <sys/conf.h>
#include <sys/signal.h>
#include <sys/file.h>
#include <sys/uio.h>
#include <sys/ioctl.h>
#include <sys/map.h>
#include <sys/proc.h>
#include <sys/user.h>
#include <sys/mman.h>
#include <sys/cred.h>
#include <sys/open.h>
#include <sys/stat.h>
#include <sys/utsname.h>
#include <sys/kmem.h>
#include <sys/cmn_err.h>
#include <sys/vnode.h>
#include <vm/page.h>
#include <vm/as.h>
#include <vm/hat.h>
#include <vm/seg.h>
#include <vm/seg_kmem.h>
#include <vm/hat_i86.h>
#include <sys/vmsystm.h>
#include <sys/ddi.h>
```



```
#include <sys/devops.h>
#include <sys/sunddi.h>
#include <sys/ddi_impldefs.h>
#include <sys/fs/snode.h>
#include <sys/pci.h>
#include <sys/modctl.h>
#include <sys/uio.h>
#include <sys/visual_io.h>
#include <sys/fbio.h>
#include <sys/ddidmareq.h>
#include <sys/tnf_probe.h>
#include <sys/kstat.h>
#include <sys/callb.h>
#include <sys/promif.h>
#include <sys/atomic.h>
#include <sys/gfx_private.h>

#ifdef __xpv
#include <sys/hypervisor.h>
#endif

/*
 * Create a kva mapping for a pa (start..start+size) with
 * the specified cache attributes (mode).
 */
gfxp_kva_t
gfxp_map_kernel_space(uint64_t start, size_t size, uint32_t mode)
{
        uint_t pgoffset;
        uint64_t base;
        pgcnt_t npages;
        caddr_t cvaddr;
        int hat_flags;
        uint_t hat_attr;
        pfn_t pfn;

        if (size == 0)
                return (0);

#ifdef __xpv
        /*
         * The hypervisor doesn't allow r/w mappings to some pages, such as
         * page tables, gdt, etc. Detect %cr3 to notify users of this interface.
         */
        if (start == mmu_ptob(mmu_btop(getcr3_pa())))
        if (start == mmu_ptob(mmu_btop(getcr3())))
                return (0);
#endif

        if (mode == GFXP_MEMORY_CACHED)
                hat_attr = HAT_STORECACHING_OK;
        else if (mode == GFXP_MEMORY_WRITECOMBINED)
                hat_attr = HAT_MERGING_OK | HAT_PLAT_NOCACHE;
        else    /* GFXP_MEMORY_UNCACHED */
                hat_attr = HAT_STRICTORDER | HAT_PLAT_NOCACHE;
        hat_flags = HAT_LOAD_LOCK;
        pgoffset = start & PAGEOFFSET;
        base = start - pgoffset;
        npages = btopr(size + pgoffset);
        cvaddr = vmem_alloc(heap_arena, ptob(npages), VM_NOSLEEP);
        if (cvaddr == NULL)
                return (NULL);

#ifdef __xpv
        ASSERT(DOMAIN_IS_INITDOMAIN(xen_info));
        pfn = xen_assign_pfn(mmu_btop(base));
```

```
125 #else
126         pfn = btop(base);
127 #endif

129         hat_devload(kas.a_hat, cvaddr, ptob(npages), pfn,
130             PROT_READ|PROT_WRITE|hat_attr, hat_flags);
131         return (cvaddr + pgoffset);
132 }
______unchanged_portion_omitted_

300 /*
301  * Like gfxp_map_kernel_space, but
302  * just the hat_devload part.
303  */
304 void
305 gfxp_load_kernel_space(uint64_t start, size_t size,
306     uint32_t mode, caddr_t cvaddr)
307 {
308         uint_t pgoffset;
309         uint64_t base;
310         pgcnt_t npages;
311         int hat_flags;
312         uint_t hat_attr;
313         pfn_t pfn;

315         if (size == 0)
316                 return;

318 #ifdef __xpv
319         /*
320          * The hypervisor doesn't allow r/w mappings to some pages, such as
321          * page tables, gdt, etc. Detect %cr3 to notify users of this interface.
322          */
323         if (start == mmu_ptob(mmu_btop(getcr3_pa())))
321         if (start == mmu_ptob(mmu_btop(getcr3())))
324                 return;
325 #endif

327         if (mode == GFXP_MEMORY_CACHED)
328                 hat_attr = HAT_STORECACHING_OK;
329         else if (mode == GFXP_MEMORY_WRITECOMBINED)
330                 hat_attr = HAT_MERGING_OK | HAT_PLAT_NOCACHE;
331         else    /* GFXP_MEMORY_UNCACHED */
332                 hat_attr = HAT_STRICTORDER | HAT_PLAT_NOCACHE;
333         hat_flags = HAT_LOAD_LOCK;

335         pgoffset = start & PAGEOFFSET;
336         base = start - pgoffset;
337         npages = btopr(size + pgoffset);

339 #ifdef __xpv
340         ASSERT(DOMAIN_IS_INITDOMAIN(xen_info));
341         pfn = xen_assign_pfn(mmu_btop(base));
342 #else
343         pfn = btop(base);
344 #endif

346         hat_devload(kas.a_hat, cvaddr, ptob(npages), pfn,
347             PROT_READ|PROT_WRITE|hat_attr, hat_flags);
348 }
______unchanged_portion_omitted_
```

```
**********************************************************
    9444 Fri Apr  6 17:25:00 2018
new/usr/src/uts/i86pc/ml/fb_swtch_src.s
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2018 Joyent, Inc.
 */


#if defined(__lint)

int fb_swtch_silence_lint = 0;

#else

#include <sys/asm_linkage.h>
#include <sys/segments.h>
#include <sys/controlregs.h>
#include <sys/machparam.h>
#include <sys/multiboot.h>
#include <sys/fastboot.h>
#include "assym.h"

/*
 * This code is to switch from 64-bit or 32-bit to protected mode.
 */

/*
 * For debugging with LEDs
 */
#define FB_OUTB_ASM(val)        \
    movb        val, %al;       \
    outb        $0x80;


#define DISABLE_PAGING                                          \
        movl    %cr4, %eax                                      ;\
        btrl    $17, %eax       /* clear PCIDE bit */           ;\
        movl    %eax, %cr4                                      ;\
        movl    %cr0, %eax                                      ;\
```



```
        btrl    $31, %eax       /* clear PG bit */              ;\
        movl    %eax, %cr0

/*
 * This macro contains common code for 64/32-bit versions of copy_sections().
 * On entry:
 *      fbf points to the fboot_file_t
 *      snum contains the number of sections
 * Registers that would be clobbered:
 *      fbs, snum, %eax, %ecx, %edi, %esi.
 * NOTE: fb_dest_pa is supposed to be in the first 1GB,
 * therefore it is safe to use 32-bit register to hold it's value
 * even for 64-bit code.
 */

#define COPY_SECT(fbf, fbs, snum)               \
        lea     FB_SECTIONS(fbf), fbs;          \
        xorl    %eax, %eax;                     \
1:      movl    FB_DEST_PA(fbf), %esi;          \
        addl    FB_SEC_OFFSET(fbs), %esi;       \
        movl    FB_SEC_PADDR(fbs), %edi;        \
        movl    FB_SEC_SIZE(fbs), %ecx;         \
        rep                                     \
          movsb;                                \
        /* Zero BSS */                          \
        movl    FB_SEC_BSS_SIZE(fbs), %ecx;     \
        rep                                     \
          stosb;                                \
        add     $FB_SECTIONS_INCR, fbs;         \
        dec     snum;                           \
        jnz     1b


        .globl  _start
_start:

        /* Disable interrupts */
        cli

#if defined(__amd64)
        /* Switch to a low memory stack */
        movq    $_start, %rsp
        addq    $FASTBOOT_STACK_OFFSET, %rsp

        /*
         * Copy from old stack to new stack
         * If the content before fi_valid gets bigger than 0x200 bytes,
         * the reserved stack size above will need to be changed.
         */
        movq    %rdi, %rsi      /* source from old stack */
        movq    %rsp, %rdi      /* destination on the new stack */
        movq    $FI_VALID, %rcx /* size to copy */
        rep
          smovb

#elif defined(__i386)
        movl    0x4(%esp), %esi /* address of fastboot info struct */

        /* Switch to a low memory stack */
        movl    $_start, %esp
        addl    $FASTBOOT_STACK_OFFSET, %esp

        /* Copy struct to stack */
        movl    %esp, %edi      /* destination on the new stack */
        movl    $FI_VALID, %ecx /* size to copy */
        rep
```

```
126           smovb

128 #endif

130 #if defined(__amd64)

132         xorl    %eax, %eax
133         xorl    %edx, %edx

135         movl    $MSR_AMD_FSBASE, %ecx
136         wrmsr

138         movl    $MSR_AMD_GSBASE, %ecx
139         wrmsr

141         movl    $MSR_AMD_KGSBASE, %ecx
142         wrmsr

144 #endif
145         /*
146          * zero out all the registers to make sure they're 16 bit clean
147          */
148 #if defined(__amd64)
149         xorq    %r8, %r8
150         xorq    %r9, %r9
151         xorq    %r10, %r10
152         xorq    %r11, %r11
153         xorq    %r12, %r12
154         xorq    %r13, %r13
155         xorq    %r14, %r14
156         xorq    %r15, %r15
157 #endif
158         xorl    %eax, %eax
159         xorl    %ebx, %ebx
160         xorl    %ecx, %ecx
161         xorl    %edx, %edx
162         xorl    %ebp, %ebp

164 #if defined(__amd64)
165         /*
166          * Load our own GDT
167          */
168         lgdt    gdt_info
169 #endif
170         /*
171          * Load our own IDT
172          */
173         lidt    idt_info

175 #if defined(__amd64)
176         /*
177          * Invalidate all TLB entries.
178          * Load temporary pagetables to copy kernel and boot-archive
179          */
180         movq    %cr4, %rax
181         andq    $_BITNOT(CR4_PGE), %rax
182         movq    %rax, %cr4
183         movq    FI_PAGETABLE_PA(%rsp), %rax
184         movq    %rax, %cr3

186         leaq    FI_FILES(%rsp), %rbx    /* offset to the files */

188         /* copy unix to final destination */
189         movq    FI_LAST_TABLE_PA(%rsp), %rsi    /* page table PA */
190         leaq    _MUL(FASTBOOT_UNIX, FI_FILES_INCR)(%rbx), %rdi
191         call    map_copy
```



```
193         /* copy boot archive to final destination */
194         movq    FI_LAST_TABLE_PA(%rsp), %rsi    /* page table PA */
195         leaq    _MUL(FASTBOOT_BOOTARCHIVE, FI_FILES_INCR)(%rbx), %rdi
196         call    map_copy

198         /* Copy sections if there are any */
199         leaq    _MUL(FASTBOOT_UNIX, FI_FILES_INCR)(%rbx), %rdi
200         movl    FB_SECTCNT(%rdi), %esi
201         cmpl    $0, %esi
202         je      1f
203         call    copy_sections
204 1:
205         /*
206          * Shut down 64 bit mode. First get into compatiblity mode.
207          */
208         movq    %rsp, %rax
209         pushq   $B32DATA_SEL
210         pushq   %rax
211         pushf
212         pushq   $B32CODE_SEL
213         pushq   $1f
214         iretq

216         .code32
217 1:
218         movl    $B32DATA_SEL, %eax
219         movw    %ax, %ss
220         movw    %ax, %ds
221         movw    %ax, %es
222         movw    %ax, %fs
223         movw    %ax, %gs

225         /*
226          * Disable long mode by:
227          * - shutting down paging (bit 31 of cr0).  This will flush the
228          *   TLBs.
229          * - turning off PCID in cr4
230          * - disabling LME (long mode enable) in EFER (extended feature reg)
231          */
232 #endif
233         DISABLE_PAGING          /* clobbers %eax */

235 #if defined(__amd64)
236         ljmp    $B32CODE_SEL, $1f
237 1:
238 #endif

240         /*
241          * Clear PGE, PAE and PSE flags as dboot expects them to be
242          * cleared.
243          */
244         movl    %cr4, %eax
245         andl    $_BITNOT(CR4_PGE | CR4_PAE | CR4_PSE), %eax
246         movl    %eax, %cr4

248 #if defined(__amd64)
249         movl    $MSR_AMD_EFER, %ecx     /* Extended Feature Enable */
250         rdmsr
251         btcl    $8, %eax                /* bit 8 Long Mode Enable bit */
252         wrmsr

254 #elif defined(__i386)
255         /*
256          * If fi_has_pae is set, re-enable paging with PAE.
257          */
```

```
258         leal    FI_FILES(%esp), %ebx    /* offset to the files */
259         movl    FI_HAS_PAE(%esp), %edi  /* need to enable paging or not */
260         cmpl    $0, %edi
261         je      paging_on               /* no need to enable paging */

263         movl    FI_LAST_TABLE_PA(%esp), %esi    /* page table PA */

265         /*
266          * Turn on PAE
267          */
268         movl    %cr4, %eax
269         orl     $CR4_PAE, %eax
270         movl    %eax, %cr4

272         /*
273          * Load top pagetable base address into cr3
274          */
275         movl    FI_PAGETABLE_PA(%esp), %eax
276         movl    %eax, %cr3

278         movl    %cr0, %eax
279         orl     $_CONST(CR0_PG | CR0_WP | CR0_AM), %eax
280         andl    $_BITNOT(CR0_NW | CR0_CD), %eax
281         movl    %eax, %cr0
282         jmp     paging_on
283 paging_on:

285         /* copy unix to final destination */
286         leal    _MUL(FASTBOOT_UNIX, FI_FILES_INCR)(%ebx), %edx
287         call    map_copy

289         /* copy boot archive to final destination */
290         leal    _MUL(FASTBOOT_BOOTARCHIVE, FI_FILES_INCR)(%ebx), %edx
291         call    map_copy

293         /* Disable paging one more time */
294         DISABLE_PAGING

296         /* Copy sections if there are any */
297         leal    _MUL(FASTBOOT_UNIX, FI_FILES_INCR)(%ebx), %edx
298         movl    FB_SECTCNT(%edx), %eax
299         cmpl    $0, %eax
300         je      1f
301         call    copy_sections
302 1:

304         /* Whatever flags we turn on we need to turn off */
305         movl    %cr4, %eax
306         andl    $_BITNOT(CR4_PAE), %eax
307         movl    %eax, %cr4
308 #endif  /* __i386 */

310 dboot_jump:
311         /* Jump to dboot */
312         movl    $DBOOT_ENTRY_ADDRESS, %edi
313         movl    FI_NEW_MBI_PA(%esp), %ebx
314         movl    $MB_BOOTLOADER_MAGIC, %eax
315         jmp     *%edi

317 #if defined(__amd64)

319         .code64
320         ENTRY_NP(copy_sections)
321         /*
322          * On entry
323          *      %rdi points to the fboot_file_t
```



```
324          *      %rsi contains number of sections
325          */
326         movq    %rdi, %rdx
327         movq    %rsi, %r9

329         COPY_SECT(%rdx, %r8, %r9)
330         ret
331         SET_SIZE(copy_sections)
______unchanged_portion_omitted_
```

```
**********************************************************
    5723 Fri Apr  6 17:25:00 2018
new/usr/src/uts/i86pc/ml/genassym.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 *
 * Copyright 2018 Joyent, Inc.
 */

#ifndef _GENASSYM
#define _GENASSYM
#endif

#define exit    kern_exit

#include <sys/types.h>
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/elf_notes.h>
#include <sys/thread.h>
#include <sys/rwlock.h>
#include <sys/proc.h>
#include <sys/cpuvar.h>
#include <sys/clock.h>
#include <sys/trap.h>
#include <sys/modctl.h>
#include <sys/traptrace.h>
#include <vm/seg.h>
#include <sys/avintr.h>
#include <sys/pic.h>
#include <sys/pit.h>
#include <sys/fp.h>
#include <sys/disp.h>
#include <sys/archsystm.h>
#include <sys/x86_archext.h>
#include <sys/sunddi.h>
#include <sys/mach_mmu.h>

#if defined(__xpv)
#include <sys/hypervisor.h>
#endif
```



```
#undef  exit            /* unhide exit, see comment above */
extern void exit(int);

/*
 * Proactively discourage anyone from referring to structures or
 * member offsets in this program.
 */
#define struct  struct...
#define OFFSET  OFFSET...

int
main(int argc, char *argv[])
{
        printf("#define\tT_AST 0x%x\n", T_AST);

        printf("#define\tLOCK_LEVEL 0x%x\n", LOCK_LEVEL);
        printf("#define\tCLOCK_LEVEL 0x%x\n", CLOCK_LEVEL);
        printf("#define\tDISP_LEVEL 0x%x\n", DISP_LEVEL);
        printf("#define\tPIL_MAX 0x%x\n", PIL_MAX);
        printf("#define\tHIGH_LEVELS 0x%x\n", HIGH_LEVELS);
        printf("#define\tCPU_INTR_ACTV_HIGH_LEVEL_MASK 0x%x\n",
            CPU_INTR_ACTV_HIGH_LEVEL_MASK);

        printf("#define\tPIC_NSEOI 0x%x\n", PIC_NSEOI);
        printf("#define\tPIC_SEOI_LVL7 0x%x\n", PIC_SEOI_LVL7);

        printf("#define\tNANOSEC 0x%llx\n", NANOSEC);
        printf("#define\tADJ_SHIFT 0x%x\n", ADJ_SHIFT);

        printf("#define\tSSLEEP 0x%x\n", SSLEEP);
        printf("#define\tSRUN 0x%x\n", SRUN);
        printf("#define\tSONPROC 0x%x\n", SONPROC);

        printf("#define\tT_INTR_THREAD 0x%x\n", T_INTR_THREAD);
        printf("#define\tFREE_THREAD 0x%x\n", TS_FREE);
        printf("#define\tTS_FREE 0x%x\n", TS_FREE);
        printf("#define\tTS_ZOMB 0x%x\n", TS_ZOMB);
        printf("#define\tTP_MSACCT 0x%x\n", TP_MSACCT);
        printf("#define\tTP_WATCHPT 0x%x\n", TP_WATCHPT);
        printf("#define\tONPROC_THREAD 0x%x\n", TS_ONPROC);

        printf("#define\tS_READ 0x%x\n", (int)S_READ);
        printf("#define\tS_WRITE 0x%x\n", (int)S_WRITE);
        printf("#define\tS_EXEC 0x%x\n", (int)S_EXEC);
        printf("#define\tS_OTHER 0x%x\n", (int)S_OTHER);

        printf("#define\tNORMALRETURN 0x%x\n", (int)NORMALRETURN);
        printf("#define\tLWP_USER 0x%x\n", LWP_USER);
        printf("#define\tLWP_SYS 0x%x\n", LWP_SYS);
        printf("#define\tLMS_USER 0x%x\n", LMS_USER);
        printf("#define\tLMS_SYSTEM 0x%x\n", LMS_SYSTEM);

        printf("#define\tSSE_MXCSR_EFLAGS 0x%x\n", SSE_MXCSR_EFLAGS);

        printf("#define\tFP_487 0x%x\n", FP_487);
        printf("#define\tFP_486 0x%x\n", FP_486);
        printf("#define\tFPU_CW_INIT 0x%x\n", FPU_CW_INIT);
        printf("#define\tFPU_EN 0x%x\n", FPU_EN);
        printf("#define\tFPU_VALID 0x%x\n", FPU_VALID);

        printf("#define\tFP_NO 0x%x\n", FP_NO);
        printf("#define\tFP_SW 0x%x\n", FP_SW);
        printf("#define\tFP_HW 0x%x\n", FP_HW);
        printf("#define\tFP_287 0x%x\n", FP_287);
        printf("#define\tFP_387 0x%x\n", FP_387);
        printf("#define\t__FP_SSE 0x%x\n", __FP_SSE);
```

```
125         printf("#define\tFP_FNSAVE 0x%x\n", FP_FNSAVE);
112         printf("#define\tFP_FXSAVE 0x%x\n", FP_FXSAVE);
113         printf("#define\tFP_XSAVE 0x%x\n", FP_XSAVE);

115         printf("#define\tAV_INT_SPURIOUS 0x%x\n", AV_INT_SPURIOUS);

117         printf("#define\tCPU_READY 0x%x\n", CPU_READY);
118         printf("#define\tCPU_QUIESCED 0x%x\n", CPU_QUIESCED);

120         printf("#define\tMCMD_PORT 0x%x\n", MCMD_PORT);
121         printf("#define\tSCMD_PORT 0x%x\n", SCMD_PORT);
122         printf("#define\tMIMR_PORT 0x%x\n", MIMR_PORT);
123         printf("#define\tSIMR_PORT 0x%x\n", SIMR_PORT);

125         printf("#define\tDMP_NOSYNC 0x%x\n", DMP_NOSYNC);

127         printf("#define\tRW_WRITER\t0x%x\n", RW_WRITER);
128         printf("#define\tRW_READER\t0x%x\n", RW_READER);

130         printf("#define\tNSYSCALL 0x%x\n", NSYSCALL);

132         printf("#define\tSE_32RVAL1 0x%x\n", SE_32RVAL1);
133         printf("#define\tSE_32RVAL2 0x%x\n", SE_32RVAL2);
134         printf("#define\tSE_64RVAL 0x%x\n", SE_64RVAL);

136         printf("#define\tMAXSYSARGS 0x%x\n", MAXSYSARGS);

138         /* Hack value just to allow clock to be kicked */
139         printf("#define\tNSEC_PER_CLOCK_TICK 0x%llx\n", NANOSEC / 100);

141         printf("#define\tNSEC_PER_COUNTER_TICK 0x%llx\n", NANOSEC / PIT_HZ);

157         printf("#define\tPITCTR0_PORT 0x%x\n", PITCTR0_PORT);
158         printf("#define\tPITCTL_PORT 0x%x\n", PITCTL_PORT);
159         printf("#define\tPIT_COUNTDOWN 0x%x\n",
160             PIT_C0 | PIT_LOADMODE | PIT_NDIVMODE);

143         printf("#define\tNBPW 0x%x\n", NBPW);

145         printf("#define\tDDI_ACCATTR_IO_SPACE 0x%x\n", DDI_ACCATTR_IO_SPACE);
146         printf("#define\tDDI_ACCATTR_DIRECT 0x%x\n", DDI_ACCATTR_DIRECT);
147         printf("#define\tDDI_ACCATTR_CPU_VADDR 0x%x\n", DDI_ACCATTR_CPU_VADDR);
148         printf("#define\tDDI_DEV_AUTOINCR 0x%x\n", DDI_DEV_AUTOINCR);

150         printf("#define\tMMU_STD_PAGESIZE 0x%x\n", (uint_t)MMU_STD_PAGESIZE);
151         printf("#define\tMMU_STD_PAGEMASK 0x%x\n", (uint_t)MMU_STD_PAGEMASK);
152         printf("#define\tFOUR_MEG 0x%x\n", (uint_t)FOUR_MEG);

154         printf("#define\tTRAPTR_NENT 0x%x\n", TRAPTR_NENT);

156         printf("#define\tCPU_DTRACE_NOFAULT 0x%x\n", CPU_DTRACE_NOFAULT);
157         printf("#define\tCPU_DTRACE_BADADDR 0x%x\n", CPU_DTRACE_BADADDR);
158         printf("#define\tCPU_DTRACE_DIVZERO 0x%x\n", CPU_DTRACE_DIVZERO);
159         printf("#define\tCPU_DTRACE_ILLOP 0x%x\n", CPU_DTRACE_ILLOP);

161         printf("#define\tMODS_NOUNLOAD 0x%x\n", MODS_NOUNLOAD);
162         printf("#define\tMODS_WEAK 0x%x\n", MODS_WEAK);
163         printf("#define\tMODS_INSTALLED 0x%x\n", MODS_INSTALLED);

165         printf("#define\tKPREEMPT_SYNC 0x%x\n", KPREEMPT_SYNC);

167 #if defined(__xpv)
168         printf("#define\tSHUTDOWN_reboot 0x%x\n", SHUTDOWN_reboot);
169         printf("#define\tSCHEDOP_block 0x%x\n", SCHEDOP_block);
170         printf("#define\tVGCF_IN_KERNEL 0x%x\n", VGCF_IN_KERNEL);
```



```
171 #endif
172         return (0);
173 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   23312 Fri Apr  6 17:25:00 2018
new/usr/src/uts/i86pc/ml/kpti_trampolines.s
9441 kmdb should stash %cr3 in kdiregs
Reviewed by: John Levon <john.levon@joyent.com>
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
```

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */
/*
 * Copyright 2018 Joyent, Inc.
 */

/*
 * This file contains the trampolines that are used by KPTI in order to be
 * able to take interrupts/trap/etc while on the "user" page table.
 *
 * We don't map the full kernel text into the user page table: instead we
 * map this one small section of trampolines (which compiles to ~13 pages).
 * These trampolines are set in the IDT always (so they will run no matter
 * whether we're on the kernel or user page table), and their primary job is to
 * pivot us to the kernel %cr3 and %rsp without ruining everything.
 *
 * All of these interrupts use the amd64 IST feature when we have KPTI enabled,
 * meaning that they will execute with their %rsp set to a known location, even
 * if we take them in the kernel.
 *
 * Over in desctbls.c (for cpu0) and mp_pc.c (other cpus) we set up the IST
 * stack to point at &cpu->cpu_m.mcpu_kpti.kf_tr_rsp. You can see the mcpu_kpti
 * (a struct kpti_frame) defined in machcpuvar.h. This struct is set up to be
 * page-aligned, and we map the page it's on into both page tables. Using a
 * struct attached to the cpu_t also means that we can use %rsp-relative
 * addressing to find anything on the cpu_t, so we don't have to touch %gs or
 * GSBASE at all on incoming interrupt trampolines (which can get pretty hairy).
 *
 * This little struct is where the CPU will push the actual interrupt frame.
 * Then, in the trampoline, we change %cr3, then figure out our destination
 * stack pointer and "pivot" to it (set %rsp and re-push the CPU's interrupt
 * frame). Then we jump to the regular ISR in the kernel text and carry on as
 * normal.
 *
 * We leave the original frame and any spilled regs behind in the kpti_frame
 * lazily until we want to return to userland. Then, we clear any spilled
 * regs from it, and overwrite the rest with our iret frame. When switching
 * this cpu to a different process (in hat_switch), we bzero the whole region to
 * make sure nothing can leak between processes.
 *
 * When we're returning back to the original place we took the interrupt later
 * (especially if it was in userland), we have to jmp back to the "return
 * trampolines" here, since when we set %cr3 back to the user value, we need to
 * be executing from code here in these shared pages and not the main kernel
 * text again. Even though it should be fine to iret directly from kernel text
 * when returning to kernel code, we make things jmp to a trampoline here just
 * for consistency.
 *
 * Note that with IST, it's very important that we always must have pivoted
```



```
 * away from the IST stack before we could possibly take any other interrupt
 * on the same IST (unless it's an end-of-the-world fault and we don't care
 * about coming back from it ever).
 *
 * This is particularly relevant to the dbgtrap/brktrap trampolines, as they
 * regularly have to happen from within trampoline code (e.g. in the sysenter
 * single-step case) and then return to the world normally. As a result, these
 * two are IST'd to their own kpti_frame right above the normal one (in the same
 * page), so they don't clobber their parent interrupt.
 *
 * To aid with debugging, we also IST the page fault (#PF/pftrap), general
 * protection fault (#GP/gptrap) and stack fault (#SS/stktrap) interrupts to
 * their own separate kpti_frame. This ensures that if we take one of these
 * due to a bug in trampoline code, we preserve the original trampoline
 * state that caused the trap.
 *
 * NMI, MCE and dblfault interrupts also are taken on their own dedicated IST
 * stacks, since they can interrupt another ISR at any time. These stacks are
 * full-sized, however, and not a little kpti_frame struct. We only set %cr3 in
 * their trampolines (and do it unconditionally), and don't bother pivoting
 * away. We're either going into the panic() path, or we're going to return
 * straight away without rescheduling, so it's fine to not be on our real
 * kthread stack (and some of the state we want to go find it with might be
 * corrupt!)
 *
 * Finally, for these "special" interrupts (NMI/MCE/double fault) we use a
 * special %cr3 value we stash here in the text (kpti_safe_cr3). We set this to
 * point at the PML4 for kas early in boot and never touch it again. Hopefully
 * it survives whatever corruption brings down the rest of the kernel!
 *
 * Syscalls are different to interrupts (at least in the SYSENTER/SYSCALL64
 * cases) in that they do not push an interrupt frame (and also have some other
 * effects). In the syscall trampolines, we assume that we can only be taking
 * the call from userland and use SWAPGS and an unconditional overwrite of %cr3.
 * We do not do any stack pivoting for syscalls (and we leave SYSENTER's
 * existing %rsp pivot untouched) -- instead we spill registers into
 * %gs:CPU_KPTI_* as we need to.
 *
 * Note that the normal %cr3 values do not cause invalidations with PCIDE - see
 * hat_switch().
 */

/*
 * The macros here mostly line up with what's in kdi_idthdl.s, too, so if you
 * fix bugs here check to see if they should be fixed there as well.
 */

#include <sys/asm_linkage.h>
#include <sys/asm_misc.h>
#include <sys/regset.h>
#include <sys/privregs.h>
#include <sys/psw.h>
#include <sys/machbrand.h>
#include <sys/param.h>

#if defined(__lint)

#include <sys/types.h>
#include <sys/thread.h>
#include <sys/systm.h>

#else   /* __lint */

#include <sys/segments.h>
#include <sys/pcb.h>
#include <sys/trap.h>
```

```
#include <sys/ftrace.h>
#include <sys/traptrace.h>
#include <sys/clock.h>
#include <sys/model.h>
#include <sys/panic.h>

#if defined(__xpv)
#include <sys/hypervisor.h>
#endif

#include "assym.h"

        .data
        DGDEF3(kpti_enable, 8, 8)
        .fill   1, 8, 1

.section ".text";
.align MMU_PAGESIZE

.global kpti_tramp_start
kpti_tramp_start:
        nop

/* This will be set by mlsetup, and then double-checked later */
.global kpti_safe_cr3
kpti_safe_cr3:
        .quad 0
        SET_SIZE(kpti_safe_cr3)

/* startup_kmem() will overwrite this */
.global kpti_kbase
kpti_kbase:
        .quad KERNELBASE
        SET_SIZE(kpti_kbase)

#define SET_KERNEL_CR3(spillreg)                \
        mov     %cr3, spillreg;                 \
        mov     spillreg, %gs:CPU_KPTI_TR_CR3;  \
        mov     %gs:CPU_KPTI_KCR3, spillreg;    \
        cmp     $0, spillreg;                   \
        je      2f;                             \
        mov     spillreg, %cr3;                 \
2:

#if DEBUG
#define SET_USER_CR3(spillreg)                  \
        mov     %cr3, spillreg;                 \
        mov     spillreg, %gs:CPU_KPTI_TR_CR3;  \
        mov     %gs:CPU_KPTI_UCR3, spillreg;    \
        mov     spillreg, %cr3
#else
#define SET_USER_CR3(spillreg)                  \
        mov     %gs:CPU_KPTI_UCR3, spillreg;    \
        mov     spillreg, %cr3
#endif

#define PIVOT_KPTI_STK(spillreg)                \
        mov     %rsp, spillreg;                 \
        mov     %gs:CPU_KPTI_RET_RSP, %rsp;     \
        pushq   T_FRAMERET_SS(spillreg);        \
        pushq   T_FRAMERET_RSP(spillreg);       \
        pushq   T_FRAMERET_RFLAGS(spillreg);    \
        pushq   T_FRAMERET_CS(spillreg);        \
        pushq   T_FRAMERET_RIP(spillreg)
```



```
#define INTERRUPT_TRAMPOLINE_P(errpush) \
        pushq   %r13;                           \
        pushq   %r14;                           \
        subq    $KPTI_R14, %rsp;                \
        /* Save current %cr3. */                \
        mov     %cr3, %r14;                     \
        mov     %r14, KPTI_TR_CR3(%rsp);        \
                                                \
        cmpw    $KCS_SEL, KPTI_CS(%rsp);        \
        je      3f;                             \
1:                                              \
        /* Change to the "kernel" %cr3 */       \
        mov     KPTI_KCR3(%rsp), %r14;          \
        cmp     $0, %r14;                       \
        je      2f;                             \
        mov     %r14, %cr3;                     \
2:                                              \
        /* Get our cpu_t in %r13 */             \
        mov     %rsp, %r13;                     \
        and     $(~(MMU_PAGESIZE - 1)), %r13;   \
        subq    $CPU_KPTI_START, %r13;          \
        /* Use top of the kthread stk */        \
        mov     CPU_THREAD(%r13), %r14;         \
        mov     T_STACK(%r14), %r14;            \
        addq    $REGSIZE+MINFRAME, %r14;        \
        jmp     4f;                             \
3:                                              \
        /* Check the %rsp in the frame. */      \
        /* Is it above kernel base? */          \
        mov     kpti_kbase, %r14;               \
        cmp     %r14, KPTI_RSP(%rsp);           \
        jb      1b;                             \
        /* Use the %rsp from the trap frame */  \
        mov     KPTI_RSP(%rsp), %r14;           \
        and     $(~0xf), %r14;                  \
4:                                              \
        mov     %rsp, %r13;                     \
        /* %r14 contains our destination stk */ \
        mov     %r14, %rsp;                     \
        pushq   KPTI_SS(%r13);                  \
        pushq   KPTI_RSP(%r13);                 \
        pushq   KPTI_RFLAGS(%r13);              \
        pushq   KPTI_CS(%r13);                  \
        pushq   KPTI_RIP(%r13);                 \
        errpush;                                \
        mov     KPTI_R14(%r13), %r14;           \
        mov     KPTI_R13(%r13), %r13

#define INTERRUPT_TRAMPOLINE_NOERR              \
        INTERRUPT_TRAMPOLINE_P(/**/)

#define INTERRUPT_TRAMPOLINE                    \
        INTERRUPT_TRAMPOLINE_P(pushq KPTI_ERR(%r13))

/*
 * This is used for all interrupts that can plausibly be taken inside another
 * interrupt and are using a kpti_frame stack (so #BP, #DB, #GP, #PF, #SS).
 *
 * We check for whether we took the interrupt while in another trampoline, in
 * which case we need to use the kthread stack.
 */
#define DBG_INTERRUPT_TRAMPOLINE_P(errpush)     \
        pushq   %r13;                           \
        pushq   %r14;                           \
        subq    $KPTI_R14, %rsp;                \
        /* Check for clobbering */              \
```

```
256         cmp     $0, KPTI_FLAG(%rsp);            \
257         je      1f;                             \
258         /* Don't worry, this totally works */   \
259         int     $8;                             \
260 1:                                              \
261         movq    $1, KPTI_FLAG(%rsp);            \
262         /* Save current %cr3. */                \
263         mov     %cr3, %r14;                     \
264         mov     %r14, KPTI_TR_CR3(%rsp);        \
265                                                 \
266         cmpw    $KCS_SEL, KPTI_CS(%rsp);        \
267         je      4f;                             \
268 2:                                              \
269         /* Change to the "kernel" %cr3 */       \
270         mov     KPTI_KCR3(%rsp), %r14;          \
271         cmp     $0, %r14;                       \
272         je      3f;                             \
273         mov     %r14, %cr3;                     \
274 3:                                              \
275         /* Get our cpu_t in %r13 */             \
276         mov     %rsp, %r13;                     \
277         and     $(~(MMU_PAGESIZE - 1)), %r13;   \
278         subq    $CPU_KPTI_START, %r13;          \
279         /* Use top of the kthread stk */        \
280         mov     CPU_THREAD(%r13), %r14;         \
281         mov     T_STACK(%r14), %r14;            \
282         addq    $REGSIZE+MINFRAME, %r14;        \
283         jmp     6f;                             \
284 4:                                              \
285         /* Check the %rsp in the frame. */      \
286         /* Is it above kernel base? */          \
287         /* If not, treat as user. */            \
288         mov     kpti_kbase, %r14;               \
289         cmp     %r14, KPTI_RSP(%rsp);           \
290         jb      2b;                             \
291         /* Is it within the kpti_frame page? */ \
292         /* If it is, treat as user interrupt */ \
293         mov     %rsp, %r13;                     \
294         and     $(~(MMU_PAGESIZE - 1)), %r13;   \
295         mov     KPTI_RSP(%rsp), %r14;           \
296         and     $(~(MMU_PAGESIZE - 1)), %r14;   \
297         cmp     %r13, %r14;                     \
298         je      2b;                             \
299         /* Were we in trampoline code? */       \
300         leaq    kpti_tramp_start, %r14;         \
301         cmp     %r14, KPTI_RIP(%rsp);           \
302         jb      5f;                             \
303         leaq    kpti_tramp_end, %r14;           \
304         cmp     %r14, KPTI_RIP(%rsp);           \
305         ja      5f;                             \
306         /* If we were, change %cr3: we might */ \
307         /* have interrupted before it did. */   \
308         mov     KPTI_KCR3(%rsp), %r14;          \
309         mov     %r14, %cr3;                     \
310 5:                                              \
311         /* Use the %rsp from the trap frame */  \
312         mov     KPTI_RSP(%rsp), %r14;           \
313         and     $(~0xf), %r14;                  \
314 6:                                              \
315         mov     %rsp, %r13;                     \
316         /* %r14 contains our destination stk */ \
317         mov     %r14, %rsp;                     \
318         pushq   KPTI_SS(%r13);                  \
319         pushq   KPTI_RSP(%r13);                 \
320         pushq   KPTI_RFLAGS(%r13);              \
321         pushq   KPTI_CS(%r13);                  \
```



```
322         pushq   KPTI_RIP(%r13);                 \
323         errpush;                                \
324         mov     KPTI_R14(%r13), %r14;           \
325         movq    $0, KPTI_FLAG(%r13);            \
326         mov     KPTI_R13(%r13), %r13

328 #define DBG_INTERRUPT_TRAMPOLINE_NOERR          \
329         DBG_INTERRUPT_TRAMPOLINE_P(/**/)

331 #define DBG_INTERRUPT_TRAMPOLINE                \
332         DBG_INTERRUPT_TRAMPOLINE_P(pushq KPTI_ERR(%r13))

334         /*
335          * These labels (_start and _end) are used by trap.c to determine if
336          * we took an interrupt like an NMI during the return process.
337          */
338 .global tr_sysc_ret_start
339 tr_sysc_ret_start:

341         /*
342          * Syscall return trampolines.
343          *
344          * These are expected to be called on the kernel %gs. tr_sysret[ql] are
345          * called after %rsp is changed back to the user value, so we have no
346          * stack to work with. tr_sysexit has a kernel stack (but has to
347          * preserve rflags, soooo).
348          */
349         ENTRY_NP(tr_sysretq)
350         cmpq    $1, kpti_enable
351         jne     1f

353         mov     %r13, %gs:CPU_KPTI_R13
354         SET_USER_CR3(%r13)
355         mov     %gs:CPU_KPTI_R13, %r13
356         /* Zero these to make sure they didn't leak from a kernel trap */
357         movq    $0, %gs:CPU_KPTI_R13
358         movq    $0, %gs:CPU_KPTI_R14
359 1:
360         swapgs
361         sysretq
362         SET_SIZE(tr_sysretq)

364         ENTRY_NP(tr_sysretl)
365         cmpq    $1, kpti_enable
366         jne     1f

368         mov     %r13, %gs:CPU_KPTI_R13
369         SET_USER_CR3(%r13)
370         mov     %gs:CPU_KPTI_R13, %r13
371         /* Zero these to make sure they didn't leak from a kernel trap */
372         movq    $0, %gs:CPU_KPTI_R13
373         movq    $0, %gs:CPU_KPTI_R14
374 1:
375         SWAPGS
376         SYSRETL
377         SET_SIZE(tr_sysretl)

379         ENTRY_NP(tr_sysexit)
380         /*
381          * Note: we want to preserve RFLAGS across this branch, since sysexit
382          * (unlike sysret above) does not restore RFLAGS for us.
383          *
384          * We still have the real kernel stack (sysexit does restore that), so
385          * we can use pushfq/popfq.
386          */
387         pushfq
```

```
389         cmpq    $1, kpti_enable
390         jne     1f

392         /* Have to pop it back off now before we change %cr3! */
393         popfq
394         mov     %r13, %gs:CPU_KPTI_R13
395         SET_USER_CR3(%r13)
396         mov     %gs:CPU_KPTI_R13, %r13
397         /* Zero these to make sure they didn't leak from a kernel trap */
398         movq    $0, %gs:CPU_KPTI_R13
399         movq    $0, %gs:CPU_KPTI_R14
400         jmp     2f
401 1:
402         popfq
403 2:
404         swapgs
405         sti
406         sysexit
407         SET_SIZE(tr_sysexit)

409 .global tr_sysc_ret_end
410 tr_sysc_ret_end:

412         /*
413          * Syscall entry trampolines.
414          */

416 #if DEBUG
417 #define MK_SYSCALL_TRAMPOLINE(isr)              \
418         ENTRY_NP(tr_/**/isr);                   \
419         swapgs;                                 \
420         mov     %r13, %gs:CPU_KPTI_R13;         \
421         mov     %cr3, %r13;                     \
422         mov     %r13, %gs:CPU_KPTI_TR_CR3;      \
423         mov     %gs:CPU_KPTI_KCR3, %r13;        \
424         mov     %r13, %cr3;                     \
425         mov     %gs:CPU_KPTI_R13, %r13;         \
426         swapgs;                                 \
427         jmp     isr;                            \
428         SET_SIZE(tr_/**/isr)
429 #else
430 #define MK_SYSCALL_TRAMPOLINE(isr)              \
431         ENTRY_NP(tr_/**/isr);                   \
432         swapgs;                                 \
433         mov     %r13, %gs:CPU_KPTI_R13;         \
434         mov     %gs:CPU_KPTI_KCR3, %r13;        \
435         mov     %r13, %cr3;                     \
436         mov     %gs:CPU_KPTI_R13, %r13;         \
437         swapgs;                                 \
438         jmp     isr;                            \
439         SET_SIZE(tr_/**/isr)
440 #endif

442         MK_SYSCALL_TRAMPOLINE(sys_syscall)
443         MK_SYSCALL_TRAMPOLINE(sys_syscall32)
444         MK_SYSCALL_TRAMPOLINE(brand_sys_syscall)
445         MK_SYSCALL_TRAMPOLINE(brand_sys_syscall32)

447         /*
448          * SYSENTER is special. The CPU is really not very helpful when it
449          * comes to preserving and restoring state with it, and as a result
450          * we have to do all of it by hand. So, since we want to preserve
451          * RFLAGS, we have to be very careful in these trampolines to not
452          * clobber any bits in it. That means no cmpqs or branches!
453          */
```



```
454         ENTRY_NP(tr_sys_sysenter)
455         swapgs
456         mov     %r13, %gs:CPU_KPTI_R13
457 #if DEBUG
458         mov     %cr3, %r13
459         mov     %r13, %gs:CPU_KPTI_TR_CR3
460 #endif
461         mov     %gs:CPU_KPTI_KCR3, %r13
462         mov     %r13, %cr3
463         mov     %gs:CPU_KPTI_R13, %r13
464         jmp     _sys_sysenter_post_swapgs
465         SET_SIZE(tr_sys_sysenter)

467         ENTRY_NP(tr_brand_sys_sysenter)
468         swapgs
469         mov     %r13, %gs:CPU_KPTI_R13
470 #if DEBUG
471         mov     %cr3, %r13
472         mov     %r13, %gs:CPU_KPTI_TR_CR3
473 #endif
474         mov     %gs:CPU_KPTI_KCR3, %r13
475         mov     %r13, %cr3
476         mov     %gs:CPU_KPTI_R13, %r13
477         jmp     _brand_sys_sysenter_post_swapgs
478         SET_SIZE(tr_brand_sys_sysenter)

480 #define MK_SYSCALL_INT_TRAMPOLINE(isr)          \
481         ENTRY_NP(tr_/**/isr);                   \
482         swapgs;                                 \
483         mov     %r13, %gs:CPU_KPTI_R13;         \
484         SET_KERNEL_CR3(%r13);                   \
485         mov     %gs:CPU_THREAD, %r13;           \
486         mov     T_STACK(%r13), %r13;            \
487         addq    $REGSIZE+MINFRAME, %r13;        \
488         mov     %r13, %rsp;                     \
489         pushq   %gs:CPU_KPTI_SS;                \
490         pushq   %gs:CPU_KPTI_RSP;               \
491         pushq   %gs:CPU_KPTI_RFLAGS;            \
492         pushq   %gs:CPU_KPTI_CS;                \
493         pushq   %gs:CPU_KPTI_RIP;               \
494         mov     %gs:CPU_KPTI_R13, %r13;         \
495         SWAPGS;                                 \
496         jmp     isr;                            \
497         SET_SIZE(tr_/**/isr)

499         MK_SYSCALL_INT_TRAMPOLINE(brand_sys_syscall_int)
500         MK_SYSCALL_INT_TRAMPOLINE(sys_syscall_int)

502         /*
503          * Interrupt/trap return trampolines
504          */

506 .global tr_intr_ret_start
507 tr_intr_ret_start:

509         ENTRY_NP(tr_iret_auto)
510         cmpq    $1, kpti_enable
511         jne     tr_iret_kernel
512         cmpw    $KCS_SEL, T_FRAMERET_CS(%rsp)
513         je      tr_iret_kernel
514         jmp     tr_iret_user
515         SET_SIZE(tr_iret_auto)

517         ENTRY_NP(tr_iret_kernel)
518         /*
519          * Yes, this does nothing extra. But this way we know if we see iret
```

```
520          * elsewhere, then we've failed to properly consider trampolines there.
521          */
522         iretq
523         SET_SIZE(tr_iret_kernel)
524
525         ENTRY_NP(tr_iret_user)
526         cmpq    $1, kpti_enable
527         jne     1f
528
529         swapgs
530         mov     %r13, %gs:CPU_KPTI_R13
531         PIVOT_KPTI_STK(%r13)
532         SET_USER_CR3(%r13)
533         mov     %gs:CPU_KPTI_R13, %r13
534         /* Zero these to make sure they didn't leak from a kernel trap */
535         movq    $0, %gs:CPU_KPTI_R13
536         movq    $0, %gs:CPU_KPTI_R14
537         swapgs
538 1:
539         iretq
540         SET_SIZE(tr_iret_user)
541
542         /*
543          * This special return trampoline is for KDI's use only (with kmdb).
544          *
545          * KDI/kmdb do not use swapgs -- they directly write the GSBASE MSR
546          * instead. This trampoline runs after GSBASE has already been changed
547          * back to the userland value (so we can't use %gs).
548          *
549          * Instead, the caller gives us a pointer to the kpti_dbg frame in %r13.
550          * The KPTI_R13 member in the kpti_dbg has already been set to what the
551          * real %r13 should be before we IRET.
552          *
553          * Additionally, KDI keeps a copy of the incoming %cr3 value when it
554          * took an interrupt, and has put that back in the kpti_dbg area for us
555          * to use, so we don't do any sniffing of %cs here. This is important
556          * so that debugging code that changes %cr3 is possible.
557          */
558         ENTRY_NP(tr_iret_kdi)
559         movq    %r14, KPTI_R14(%r13)    /* %r14 has to be preserved by us */
560
561         movq    %rsp, %r14      /* original %rsp is pointing at IRET frame */
562         leaq    KPTI_TOP(%r13), %rsp
563         pushq   T_FRAMERET_SS(%r14)
564         pushq   T_FRAMERET_RSP(%r14)
565         pushq   T_FRAMERET_RFLAGS(%r14)
566         pushq   T_FRAMERET_CS(%r14)
567         pushq   T_FRAMERET_RIP(%r14)
568
569         movq    KPTI_TR_CR3(%r13), %r14
570         movq    %r14, %cr3
571
572         movq    KPTI_R14(%r13), %r14
573         movq    KPTI_R13(%r13), %r13    /* preserved by our caller */
574
575         iretq
576         SET_SIZE(tr_iret_kdi)
577
578 .global tr_intr_ret_end
579 tr_intr_ret_end:
580
581         /*
582          * Interrupt/trap entry trampolines
583          */
584
585         /* CPU pushed an error code, and ISR wants one */
```



```
586 #define MK_INTR_TRAMPOLINE(isr)                 \
587         ENTRY_NP(tr_/**/isr);                   \
588         INTERRUPT_TRAMPOLINE;                   \
589         jmp     isr;                            \
590         SET_SIZE(tr_/**/isr)
591
592         /* CPU didn't push an error code, and ISR doesn't want one */
593 #define MK_INTR_TRAMPOLINE_NOERR(isr)           \
594         ENTRY_NP(tr_/**/isr);                   \
595         push    $0;                             \
596         INTERRUPT_TRAMPOLINE_NOERR;             \
597         jmp     isr;                            \
598         SET_SIZE(tr_/**/isr)
599
600         /* CPU pushed an error code, and ISR wants one */
601 #define MK_DBG_INTR_TRAMPOLINE(isr)     \
602         ENTRY_NP(tr_/**/isr);                   \
603         DBG_INTERRUPT_TRAMPOLINE;               \
604         jmp     isr;                            \
605         SET_SIZE(tr_/**/isr)
606
607         /* CPU didn't push an error code, and ISR doesn't want one */
608 #define MK_DBG_INTR_TRAMPOLINE_NOERR(isr)       \
609         ENTRY_NP(tr_/**/isr);                   \
610         push    $0;                             \
611         DBG_INTERRUPT_TRAMPOLINE_NOERR;         \
612         jmp     isr;                            \
613         SET_SIZE(tr_/**/isr)
614
615
616         MK_INTR_TRAMPOLINE_NOERR(div0trap)
617         MK_DBG_INTR_TRAMPOLINE_NOERR(dbgtrap)
618         MK_DBG_INTR_TRAMPOLINE_NOERR(brktrap)
619         MK_INTR_TRAMPOLINE_NOERR(ovflotrap)
620         MK_INTR_TRAMPOLINE_NOERR(boundstrap)
621         MK_INTR_TRAMPOLINE_NOERR(invoptrap)
622         MK_INTR_TRAMPOLINE_NOERR(ndptrap)
623         MK_INTR_TRAMPOLINE(invtsstrap)
624         MK_INTR_TRAMPOLINE(segnptrap)
625         MK_DBG_INTR_TRAMPOLINE(stktrap)
626         MK_DBG_INTR_TRAMPOLINE(gptrap)
627         MK_DBG_INTR_TRAMPOLINE(pftrap)
628         MK_INTR_TRAMPOLINE_NOERR(resvtrap)
629         MK_INTR_TRAMPOLINE_NOERR(ndperr)
630         MK_INTR_TRAMPOLINE(achktrap)
631         MK_INTR_TRAMPOLINE_NOERR(xmtrap)
632         MK_INTR_TRAMPOLINE_NOERR(invaltrap)
633         MK_INTR_TRAMPOLINE_NOERR(fasttrap)
634         MK_INTR_TRAMPOLINE_NOERR(dtrace_ret)
635
636         /*
637          * These are special because they can interrupt other traps, and
638          * each other. We don't need to pivot their stacks, because they have
639          * dedicated IST stack space, but we need to change %cr3.
640          */
641         ENTRY_NP(tr_nmiint)
642         pushq   %r13
643         mov     kpti_safe_cr3, %r13
644         mov     %r13, %cr3
645         popq    %r13
646         jmp     nmiint
647         SET_SIZE(tr_nmiint)
648
649 #if !defined(__xpv)
650         ENTRY_NP(tr_syserrtrap)
651         /*
```

```
652          * If we got here we should always have a zero error code pushed.
653          * The INT $0x8 instr doesn't seem to push one, though, which we use
654          * as an emergency panic in the other trampolines. So adjust things
655          * here.
656          */
657         cmpq    $0, (%rsp)
658         je      1f
659         pushq   $0
660 1:
661         pushq   %r13
662         mov     kpti_safe_cr3, %r13
663         mov     %r13, %cr3
664         popq    %r13
665         jmp     syserrtrap
666         SET_SIZE(tr_syserrtrap)
667 #endif
668
669         ENTRY_NP(tr_mcetrap)
670         pushq   %r13
671         mov     kpti_safe_cr3, %r13
672         mov     %r13, %cr3
673         popq    %r13
674         jmp     mcetrap
675         SET_SIZE(tr_mcetrap)
676
677         /*
678          * Interrupts start at 32
679          */
680 #define MKIVCT(n)                       \
681         ENTRY_NP(tr_ivct/**/n)          \
682         push    $0;                     \
683         INTERRUPT_TRAMPOLINE;           \
684         push    $n - 0x20;              \
685         jmp     cmnint;                 \
686         SET_SIZE(tr_ivct/**/n)
687
688         MKIVCT(32);     MKIVCT(33);     MKIVCT(34);     MKIVCT(35);
689         MKIVCT(36);     MKIVCT(37);     MKIVCT(38);     MKIVCT(39);
690         MKIVCT(40);     MKIVCT(41);     MKIVCT(42);     MKIVCT(43);
691         MKIVCT(44);     MKIVCT(45);     MKIVCT(46);     MKIVCT(47);
692         MKIVCT(48);     MKIVCT(49);     MKIVCT(50);     MKIVCT(51);
693         MKIVCT(52);     MKIVCT(53);     MKIVCT(54);     MKIVCT(55);
694         MKIVCT(56);     MKIVCT(57);     MKIVCT(58);     MKIVCT(59);
695         MKIVCT(60);     MKIVCT(61);     MKIVCT(62);     MKIVCT(63);
696         MKIVCT(64);     MKIVCT(65);     MKIVCT(66);     MKIVCT(67);
697         MKIVCT(68);     MKIVCT(69);     MKIVCT(70);     MKIVCT(71);
698         MKIVCT(72);     MKIVCT(73);     MKIVCT(74);     MKIVCT(75);
699         MKIVCT(76);     MKIVCT(77);     MKIVCT(78);     MKIVCT(79);
700         MKIVCT(80);     MKIVCT(81);     MKIVCT(82);     MKIVCT(83);
701         MKIVCT(84);     MKIVCT(85);     MKIVCT(86);     MKIVCT(87);
702         MKIVCT(88);     MKIVCT(89);     MKIVCT(90);     MKIVCT(91);
703         MKIVCT(92);     MKIVCT(93);     MKIVCT(94);     MKIVCT(95);
704         MKIVCT(96);     MKIVCT(97);     MKIVCT(98);     MKIVCT(99);
705         MKIVCT(100);    MKIVCT(101);    MKIVCT(102);    MKIVCT(103);
706         MKIVCT(104);    MKIVCT(105);    MKIVCT(106);    MKIVCT(107);
707         MKIVCT(108);    MKIVCT(109);    MKIVCT(110);    MKIVCT(111);
708         MKIVCT(112);    MKIVCT(113);    MKIVCT(114);    MKIVCT(115);
709         MKIVCT(116);    MKIVCT(117);    MKIVCT(118);    MKIVCT(119);
710         MKIVCT(120);    MKIVCT(121);    MKIVCT(122);    MKIVCT(123);
711         MKIVCT(124);    MKIVCT(125);    MKIVCT(126);    MKIVCT(127);
712         MKIVCT(128);    MKIVCT(129);    MKIVCT(130);    MKIVCT(131);
713         MKIVCT(132);    MKIVCT(133);    MKIVCT(134);    MKIVCT(135);
714         MKIVCT(136);    MKIVCT(137);    MKIVCT(138);    MKIVCT(139);
715         MKIVCT(140);    MKIVCT(141);    MKIVCT(142);    MKIVCT(143);
716         MKIVCT(144);    MKIVCT(145);    MKIVCT(146);    MKIVCT(147);
717         MKIVCT(148);    MKIVCT(149);    MKIVCT(150);    MKIVCT(151);
```



```
718         MKIVCT(152);    MKIVCT(153);    MKIVCT(154);    MKIVCT(155);
719         MKIVCT(156);    MKIVCT(157);    MKIVCT(158);    MKIVCT(159);
720         MKIVCT(160);    MKIVCT(161);    MKIVCT(162);    MKIVCT(163);
721         MKIVCT(164);    MKIVCT(165);    MKIVCT(166);    MKIVCT(167);
722         MKIVCT(168);    MKIVCT(169);    MKIVCT(170);    MKIVCT(171);
723         MKIVCT(172);    MKIVCT(173);    MKIVCT(174);    MKIVCT(175);
724         MKIVCT(176);    MKIVCT(177);    MKIVCT(178);    MKIVCT(179);
725         MKIVCT(180);    MKIVCT(181);    MKIVCT(182);    MKIVCT(183);
726         MKIVCT(184);    MKIVCT(185);    MKIVCT(186);    MKIVCT(187);
727         MKIVCT(188);    MKIVCT(189);    MKIVCT(190);    MKIVCT(191);
728         MKIVCT(192);    MKIVCT(193);    MKIVCT(194);    MKIVCT(195);
729         MKIVCT(196);    MKIVCT(197);    MKIVCT(198);    MKIVCT(199);
730         MKIVCT(200);    MKIVCT(201);    MKIVCT(202);    MKIVCT(203);
731         MKIVCT(204);    MKIVCT(205);    MKIVCT(206);    MKIVCT(207);
732         MKIVCT(208);    MKIVCT(209);    MKIVCT(210);    MKIVCT(211);
733         MKIVCT(212);    MKIVCT(213);    MKIVCT(214);    MKIVCT(215);
734         MKIVCT(216);    MKIVCT(217);    MKIVCT(218);    MKIVCT(219);
735         MKIVCT(220);    MKIVCT(221);    MKIVCT(222);    MKIVCT(223);
736         MKIVCT(224);    MKIVCT(225);    MKIVCT(226);    MKIVCT(227);
737         MKIVCT(228);    MKIVCT(229);    MKIVCT(230);    MKIVCT(231);
738         MKIVCT(232);    MKIVCT(233);    MKIVCT(234);    MKIVCT(235);
739         MKIVCT(236);    MKIVCT(237);    MKIVCT(238);    MKIVCT(239);
740         MKIVCT(240);    MKIVCT(241);    MKIVCT(242);    MKIVCT(243);
741         MKIVCT(244);    MKIVCT(245);    MKIVCT(246);    MKIVCT(247);
742         MKIVCT(248);    MKIVCT(249);    MKIVCT(250);    MKIVCT(251);
743         MKIVCT(252);    MKIVCT(253);    MKIVCT(254);    MKIVCT(255);
744
745         /*
746          * We're PCIDE, but we don't have INVPCID.  The only way to invalidate a
747          * PCID other than the current one, then, is to load its cr3 then
748          * invlpg.  But loading kf_user_cr3 means we can longer access our
749          * caller's text mapping (or indeed, its stack).  So this little helper
750          * has to live within our trampoline text region.
751          *
752          * Called as tr_mmu_flush_user_range(addr, len, pgsz, cr3)
753          */
754         ENTRY_NP(tr_mmu_flush_user_range)
755         push    %rbx
756         /* When we read cr3, it never has the NOINVL bit set. */
757         mov     %cr3, %rax
758         movq    $CR3_NOINVL_BIT, %rbx
759         orq     %rbx, %rax
760
761         mov     %rcx, %cr3
762         add     %rdi, %rsi
763 .align  ASM_ENTRY_ALIGN
764 1:
765         invlpg  (%rdi)
766         add     %rdx, %rdi
767         cmp     %rsi, %rdi
768         jb      1b
769         mov     %rax, %cr3
770         pop     %rbx
771         retq
772         SET_SIZE(tr_mmu_flush_user_range)
773
774 .align MMU_PAGESIZE
775 .global kpti_tramp_end
776 kpti_tramp_end:
777         nop
778
779 #endif  /* __lint */
```

```
**********************************************************
   52082 Fri Apr  6 17:25:00 2018
new/usr/src/uts/i86pc/ml/locore.s
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
  24  */
  25 /*
  26  * Copyright (c) 2018 Joyent, Inc.
  26  * Copyright (c) 2016, Joyent, Inc. All rights reserved.
  27  */

  29 /*      Copyright (c) 1990, 1991 UNIX System Laboratories, Inc. */
  30 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989, 1990 AT&T   */
  31 /*        All Rights Reserved                                   */

  33 /*      Copyright (c) 1987, 1988 Microsoft Corporation          */
  34 /*        All Rights Reserved                                   */


  37 #include <sys/asm_linkage.h>
  38 #include <sys/asm_misc.h>
  39 #include <sys/regset.h>
  40 #include <sys/privregs.h>
  41 #include <sys/psw.h>
  42 #include <sys/reboot.h>
  43 #include <sys/x86_archext.h>
  44 #include <sys/machparam.h>

  46 #if defined(__lint)

  48 #include <sys/types.h>
  49 #include <sys/thread.h>
  50 #include <sys/systm.h>
  51 #include <sys/lgrp.h>
  52 #include <sys/regset.h>
  53 #include <sys/link.h>
  54 #include <sys/bootconf.h>
  55 #include <sys/bootsvcs.h>

  57 #else   /* __lint */
```



```
  59 #include <sys/segments.h>
  60 #include <sys/pcb.h>
  61 #include <sys/trap.h>
  62 #include <sys/ftrace.h>
  63 #include <sys/traptrace.h>
  64 #include <sys/clock.h>
  65 #include <sys/cmn_err.h>
  66 #include <sys/pit.h>
  67 #include <sys/panic.h>

  69 #if defined(__xpv)
  70 #include <sys/hypervisor.h>
  71 #endif

  73 #include "assym.h"

  75 /*
  76  * Our assumptions:
  77  *      - We are running in protected-paged mode.
  78  *      - Interrupts are disabled.
  79  *      - The GDT and IDT are the callers; we need our copies.
  80  *      - The kernel's text, initialized data and bss are mapped.
  81  *
  82  * Our actions:
  83  *      - Save arguments
  84  *      - Initialize our stack pointer to the thread 0 stack (t0stack)
  85  *        and leave room for a phony "struct regs".
  86  *      - Our GDT and IDT need to get munged.
  87  *      - Since we are using the boot's GDT descriptors, we need
  88  *        to copy them into our GDT before we switch to ours.
  89  *      - We start using our GDT by loading correct values in the
  90  *        selector registers (cs=KCS_SEL, ds=es=ss=KDS_SEL, fs=KFS_SEL,
  91  *        gs=KGS_SEL).
  92  *      - The default LDT entry for syscall is set.
  93  *      - We load the default LDT into the hardware LDT register.
  94  *      - We load the default TSS into the hardware task register.
  95  *      - Check for cpu type, i.e. 486 vs. P5 vs. P6 etc.
  96  *      - mlsetup(%esp) gets called.
  97  *      - We change our appearance to look like the real thread 0.
  98  *        (NOTE: making ourselves to be a real thread may be a noop)
  99  *      - main() gets called.  (NOTE: main() never returns).
 100  *
 101  * NOW, the real code!
 102  */
 103         /*
 104          * The very first thing in the kernel's text segment must be a jump
 105          * to the os/fakebop.c startup code.
 106          */
 107         .text
 108         jmp     _start

 110         /*
 111          * Globals:
 112          */
 113         .globl  _locore_start
 114         .globl  mlsetup
 115         .globl  main
 116         .globl  panic
 117         .globl  t0stack
 118         .globl  t0
 119         .globl  sysp
 120         .globl  edata

 122         /*
 123          * call back into boot - sysp (bootsvcs.h) and bootops (bootconf.h)
 124          */
```

```
 125         .globl  bootops
 126         .globl  bootopsp

 128         /*
 129          * NOTE: t0stack should be the first thing in the data section so that
 130          * if it ever overflows, it will fault on the last kernel text page.
 131          */
 132         .data
 133         .comm   t0stack, DEFAULTSTKSZ, 32
 134         .comm   t0, 4094, 32

 136 #endif  /* __lint */


 139 #if defined(__amd64)

 141 #if defined(__lint)

 143 /* ARGSUSED */
 144 void
 145 _locore_start(struct boot_syscalls *sysp, ulong_t rsi, struct bootops *bop)
 146 {}

 148 #else   /* __lint */

 150         /*
 151          * kobj_init() vectors us back to here with (note) a slightly different
 152          * set of arguments than _start is given (see lint prototypes above).
 153          *
 154          * XXX  Make this less vile, please.
 155          */
 156         ENTRY_NP(_locore_start)

 158         /*
 159          * %rdi = boot services (should die someday)
 160          * %rdx = bootops
 161          * end
 162          */

 164         leaq    edata(%rip), %rbp       /* reference edata for ksyms */
 165         movq    $0, (%rbp)              /* limit stack back trace */

 167         /*
 168          * Initialize our stack pointer to the thread 0 stack (t0stack)
 169          * and leave room for a "struct regs" for lwp0.  Note that the
 170          * stack doesn't actually align to a 16-byte boundary until just
 171          * before we call mlsetup because we want to use %rsp to point at
 172          * our regs structure.
 173          */
 174         leaq    t0stack(%rip), %rsp
 175         addq    $_CONST(DEFAULTSTKSZ - REGSIZE), %rsp
 176 #if (REGSIZE & 15) == 0
 177         subq    $8, %rsp
 178 #endif
 179         /*
 180          * Save call back for special x86 boot services vector
 181          */
 182         movq    %rdi, sysp(%rip)

 184         movq    %rdx, bootops(%rip)             /* save bootops */
 185         movq    $bootops, bootopsp(%rip)

 187         /*
 188          * Save arguments and flags, if only for debugging ..
 189          */
 190         movq    %rdi, REGOFF_RDI(%rsp)
```



```
 191         movq    %rsi, REGOFF_RSI(%rsp)
 192         movq    %rdx, REGOFF_RDX(%rsp)
 193         movq    %rcx, REGOFF_RCX(%rsp)
 194         movq    %r8, REGOFF_R8(%rsp)
 195         movq    %r9, REGOFF_R9(%rsp)
 196         pushf
 197         popq    %r11
 198         movq    %r11, REGOFF_RFL(%rsp)

 200 #if !defined(__xpv)
 201         /*
 202          * Enable write protect and alignment check faults.
 203          */
 204         movq    %cr0, %rax
 205         orq     $_CONST(CR0_WP|CR0_AM), %rax
 206         andq    $_BITNOT(CR0_WT|CR0_CE), %rax
 207         movq    %rax, %cr0
 208 #endif  /* __xpv */

 210         /*
 211          * (We just assert this works by virtue of being here)
 212          */
 213         bts     $X86FSET_CPUID, x86_featureset(%rip)

 215         /*
 216          * mlsetup() gets called with a struct regs as argument, while
 217          * main takes no args and should never return.
 218          */
 219         xorl    %ebp, %ebp
 220         movq    %rsp, %rdi
 221         pushq   %rbp
 222         /* (stack pointer now aligned on 16-byte boundary right here) */
 223         movq    %rsp, %rbp
 224         call    mlsetup
 225         call    main
 226         /* NOTREACHED */
 227         leaq    __return_from_main(%rip), %rdi
 228         xorl    %eax, %eax
 229         call    panic
 230         SET_SIZE(_locore_start)
______unchanged_portion_omitted_

1075 #endif  /* __lint */
1076 #endif  /* !__amd64 */


1079 /*
1080  *  For stack layout, see privregs.h
1081  *  When cmntrap gets called, the error code and trap number have been pushed.
1082  *  When cmntrap_pushed gets called, the entire struct regs has been pushed.
1083  */

1085 #if defined(__lint)

1087 /* ARGSUSED */
1088 void
1089 cmntrap()
1090 {}

1092 #else   /* __lint */

1094         .globl  trap            /* C handler called below */

1096 #if defined(__amd64)

1098         ENTRY_NP2(cmntrap, _cmntrap)
```

```
1100         INTR_PUSH

1102         ALTENTRY(cmntrap_pushed)

1104         movq    %rsp, %rbp

1106         /*
1107          * - if this is a #pf i.e. T_PGFLT, %r15 is live
1108          *   and contains the faulting address i.e. a copy of %cr2
1109          *
1110          * - if this is a #db i.e. T_SGLSTP, %r15 is live
1111          *   and contains the value of %db6
1112          */

1114         TRACE_PTR(%rdi, %rbx, %ebx, %rcx, $TT_TRAP) /* Uses labels 8 and 9 */
1115         TRACE_REGS(%rdi, %rsp, %rbx, %rcx)      /* Uses label 9 */
1116         TRACE_STAMP(%rdi)               /* Clobbers %eax, %edx, uses 9 */

1118         /*
1119          * We must first check if DTrace has set its NOFAULT bit.  This
1120          * regrettably must happen before the trap stack is recorded, because
1121          * this requires a call to getpcstack() and may induce recursion if an
1122          * fbt::getpcstack: enabling is inducing the bad load.
1123          */
1124         movl    %gs:CPU_ID, %eax
1125         shlq    $CPU_CORE_SHIFT, %rax
1126         leaq    cpu_core(%rip), %r8
1127         addq    %r8, %rax
1128         movw    CPUC_DTRACE_FLAGS(%rax), %cx
1129         testw   $CPU_DTRACE_NOFAULT, %cx
1130         jnz     .dtrace_induced

1132         TRACE_STACK(%rdi)

1134         movq    %rbp, %rdi
1135         movq    %r15, %rsi
1136         movl    %gs:CPU_ID, %edx

1138         /*
1139          * We know that this isn't a DTrace non-faulting load; we can now safely
1140          * reenable interrupts.  (In the case of pagefaults, we enter through an
1141          * interrupt gate.)
1142          */
1143         ENABLE_INTR_FLAGS

1145         call    trap            /* trap(rp, addr, cpuid) handles all traps */
1146         jmp     _sys_rtt

1148 .dtrace_induced:
1149         cmpw    $KCS_SEL, REGOFF_CS(%rbp)       /* test CS for user-mode trap */
1150         jne     3f                              /* if from user, panic */

1152         cmpl    $T_PGFLT, REGOFF_TRAPNO(%rbp)
1153         je      1f

1155         cmpl    $T_GPFLT, REGOFF_TRAPNO(%rbp)
1156         je      0f

1158         cmpl    $T_ILLINST, REGOFF_TRAPNO(%rbp)
1159         je      0f

1161         cmpl    $T_ZERODIV, REGOFF_TRAPNO(%rbp)
1162         jne     4f                              /* if not PF/GP/UD/DE, panic */

1164         orw     $CPU_DTRACE_DIVZERO, %cx
```



```
1165         movw    %cx, CPUC_DTRACE_FLAGS(%rax)
1166         jmp     2f

1168         /*
1169          * If we've taken a GPF, we don't (unfortunately) have the address that
1170          * induced the fault.  So instead of setting the fault to BADADDR,
1171          * we'll set the fault to ILLOP.
1172          */
1173 0:
1174         orw     $CPU_DTRACE_ILLOP, %cx
1175         movw    %cx, CPUC_DTRACE_FLAGS(%rax)
1176         jmp     2f
1177 1:
1178         orw     $CPU_DTRACE_BADADDR, %cx
1179         movw    %cx, CPUC_DTRACE_FLAGS(%rax)    /* set fault to bad addr */
1180         movq    %r15, CPUC_DTRACE_ILLVAL(%rax)
1181                                                 /* fault addr is illegal value */
1182 2:
1183         movq    REGOFF_RIP(%rbp), %rdi
1184         movq    %rdi, %r12
1185         call    dtrace_instr_size
1186         addq    %rax, %r12
1187         movq    %r12, REGOFF_RIP(%rbp)
1188         INTR_POP
1189         jmp     tr_iret_auto
1189         IRET
1190         /*NOTREACHED*/
1191 3:
1192         leaq    dtrace_badflags(%rip), %rdi
1193         xorl    %eax, %eax
1194         call    panic
1195 4:
1196         leaq    dtrace_badtrap(%rip), %rdi
1197         xorl    %eax, %eax
1198         call    panic
1199         SET_SIZE(cmntrap)
_______unchanged_portion_omitted_

1484 #endif  /* __i386 */

1486 #endif  /* __lint */

1488 /*
1489  * Return from _sys_trap routine.
1490  */

1492 #if defined(__lint)

1494 void
1495 lwp_rtt_initial(void)
1496 {}

1498 void
1499 lwp_rtt(void)
1500 {}

1502 void
1503 _sys_rtt(void)
1504 {}

1506 #else   /* __lint */

1508 #if defined(__amd64)

1510         ENTRY_NP(lwp_rtt_initial)
1511         movq    %gs:CPU_THREAD, %r15
```

```
1512         movq    T_STACK(%r15), %rsp     /* switch to the thread stack */
1513         movq    %rsp, %rbp
1514         call    __dtrace_probe___proc_start
1515         jmp     _lwp_rtt

1517         ENTRY_NP(lwp_rtt)

1519         /*
1520          * r14  lwp
1521          * rdx  lwp->lwp_procp
1522          * r15  curthread
1523          */

1525         movq    %gs:CPU_THREAD, %r15
1526         movq    T_STACK(%r15), %rsp     /* switch to the thread stack */
1527         movq    %rsp, %rbp
1528 _lwp_rtt:
1529         call    __dtrace_probe___proc_lwp__start
1530         movq    %gs:CPU_LWP, %r14
1531         movq    LWP_PROCP(%r14), %rdx

1533         /*
1534          * XX64 Is the stack misaligned correctly at this point?
1535          *      If not, we need to do a push before calling anything ..
1536          */

1538 #if defined(DEBUG)
1539         /*
1540          * If we were to run lwp_savectx at this point -without-
1541          * pcb_rupdate being set to 1, we'd end up sampling the hardware
1542          * state left by the previous running lwp, rather than setting
1543          * the values requested by the lwp creator.  Bad.
1544          */
1545         testb   $0x1, PCB_RUPDATE(%r14)
1546         jne     1f
1547         leaq    _no_pending_updates(%rip), %rdi
1548         movl    $__LINE__, %esi
1549         movq    %r14, %rdx
1550         xorl    %eax, %eax
1551         call    panic
1552 _no_pending_updates:
1553         .string "locore.s:%d lwp_rtt(lwp %p) but pcb_rupdate != 1"
1554 1:
1555 #endif

1557         /*
1558          * If agent lwp, clear %fs and %gs
1559          */
1560         cmpq    %r15, P_AGENTTP(%rdx)
1561         jne     1f
1562         xorl    %ecx, %ecx
1563         movq    %rcx, REGOFF_FS(%rsp)
1564         movq    %rcx, REGOFF_GS(%rsp)
1565         movw    %cx, LWP_PCB_FS(%r14)
1566         movw    %cx, LWP_PCB_GS(%r14)
1567 1:
1568         call    dtrace_systrace_rtt
1569         movq    REGOFF_RDX(%rsp), %rsi
1570         movq    REGOFF_RAX(%rsp), %rdi
1571         call    post_syscall            /* post_syscall(rval1, rval2) */

1573         /*
1574          * set up to take fault on first use of fp
1575          */
1576         STTS(%rdi)
```



```
1578         /*
1579          * XXX - may want a fast path that avoids sys_rtt_common in the
1580          * most common case.
1581          */
1582         ALTENTRY(_sys_rtt)
1583         CLI(%rax)                       /* disable interrupts */
1584         ALTENTRY(_sys_rtt_ints_disabled)
1585         movq    %rsp, %rdi              /* pass rp to sys_rtt_common */
1586         call    sys_rtt_common          /* do common sys_rtt tasks */
1587         testq   %rax, %rax              /* returning to userland? */
1588         jz      sr_sup

1590         /*
1591          * Return to user
1592          */
1593         ASSERT_UPCALL_MASK_IS_SET
1594         cmpw    $UCS_SEL, REGOFF_CS(%rsp) /* test for native (64-bit) lwp? */
1595         je      sys_rtt_syscall

1597         /*
1598          * Return to 32-bit userland
1599          */
1600         ALTENTRY(sys_rtt_syscall32)
1601         USER32_POP
1602         jmp     tr_iret_user
1602         IRET
1603         /*NOTREACHED*/

1605         ALTENTRY(sys_rtt_syscall)
1606         /*
1607          * Return to 64-bit userland
1608          */
1609         USER_POP
1610         ALTENTRY(nopop_sys_rtt_syscall)
1611         jmp     tr_iret_user
1611         IRET
1612         /*NOTREACHED*/
1613         SET_SIZE(nopop_sys_rtt_syscall)

1615         /*
1616          * Return to supervisor
1617          * NOTE: to make the check in trap() that tests if we are executing
1618          * segment register fixup/restore code work properly, sr_sup MUST be
1619          * after _sys_rtt .
1620          */
1621         ALTENTRY(sr_sup)
1622         /*
1623          * Restore regs before doing iretq to kernel mode
1624          */
1625         INTR_POP
1626         jmp     tr_iret_kernel
1626         IRET
1627         .globl  _sys_rtt_end
1628 _sys_rtt_end:
1629         /*NOTREACHED*/
1630         SET_SIZE(sr_sup)
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   15320 Fri Apr  6 17:25:00 2018
new/usr/src/uts/i86pc/ml/mpcore.s
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 *
 * Copyright 2018 Joyent, Inc.
 */

#include <sys/asm_linkage.h>
#include <sys/asm_misc.h>
#include <sys/regset.h>
#include <sys/privregs.h>
#include <sys/x86_archext.h>

#if !defined(__lint)
#include <sys/segments.h>
#include "assym.h"
#endif

/*
 *      Our assumptions:
 *              - We are running in real mode.
 *              - Interrupts are disabled.
 *              - Selectors are equal (cs == ds == ss) for all real mode code
 *              - The GDT, IDT, ktss and page directory has been built for us
 *
 *      Our actions:
 *      Start CPU:
 *              - We start using our GDT by loading correct values in the
 *                selector registers (cs=KCS_SEL, ds=es=ss=KDS_SEL, fs=KFS_SEL,
 *                gs=KGS_SEL).
 *              - We change over to using our IDT.
 *              - We load the default LDT into the hardware LDT register.
 *              - We load the default TSS into the hardware task register.
 *              - call mp_startup(void) indirectly through the T_PC
 *      Stop CPU:
 *              - Put CPU into halted state with interrupts disabled
 *
 *
 */

#if defined(__lint)

void
real_mode_start_cpu(void)
{}

void
real_mode_stop_cpu_stage1(void)
{}

void
real_mode_stop_cpu_stage2(void)
{}

#else   /* __lint */

#if defined(__amd64)

        ENTRY_NP(real_mode_start_cpu)

        /*
         * NOTE:  The GNU assembler automatically does the right thing to
         *        generate data size operand prefixes based on the code size
         *        generation mode (e.g. .code16, .code32, .code64) and as such
         *        prefixes need not be used on instructions EXCEPT in the case
         *        of address prefixes for code for which the reference is not
         *        automatically of the default operand size.
         */
        .code16
        cli
        movw            %cs, %ax
        movw            %ax, %ds        /* load cs into ds */
        movw            %ax, %ss        /* and into ss */

        /*
         * Helps in debugging by giving us the fault address.
         *
         * Remember to patch a hlt (0xf4) at cmntrap to get a good stack.
         */
        movl            $0xffc, %esp
        movl            %cr0, %eax

        /*
         * Enable protected-mode, write protect, and alignment mask
         */
        orl             $(CR0_PE|CR0_WP|CR0_AM), %eax
        movl            %eax, %cr0

        /*
         * Do a jmp immediately after writing to cr0 when enabling protected
         * mode to clear the real mode prefetch queue (per Intel's docs)
         */
        jmp             pestart

pestart:
        /*
         * 16-bit protected mode is now active, so prepare to turn on long
         * mode.
         *
         * Note that we currently assume that if we're attempting to run a
         * kernel compiled with (__amd64) #defined, the target CPU has long
         * mode support.
         */
```

```
#if 0
        /*
         * If there's a chance this might not be true, the following test should
         * be done, with the no_long_mode branch then doing something
         * appropriate:
         */

        movl            $0x80000000, %eax       /* get largest extended CPUID */
        cpuid
        cmpl            $0x80000000, %eax       /* check if > 0x80000000 */
        jbe             no_long_mode            /* nope, no long mode */
        movl            $0x80000001, %eax
        cpuid                                   /* get extended feature flags */
        btl             $29, %edx               /* check for long mode */
        jnc             no_long_mode            /* long mode not supported */
#endif

        /*
         * Add any initial cr4 bits
         */
        movl            %cr4, %eax
        addr32 orl      CR4OFF, %eax

        /*
         * Enable PAE mode (CR4.PAE)
         */
        orl             $CR4_PAE, %eax
        movl            %eax, %cr4

        /*
         * Point cr3 to the 64-bit long mode page tables.
         *
         * Note that these MUST exist in 32-bit space, as we don't have
         * a way to load %cr3 with a 64-bit base address for the page tables
         * until the CPU is actually executing in 64-bit long mode.
         */
        addr32 movl     CR3OFF, %eax
        movl            %eax, %cr3

        /*
         * Set long mode enable in EFER (EFER.LME = 1)
         */
        movl    $MSR_AMD_EFER, %ecx
        rdmsr
        orl     $AMD_EFER_LME, %eax
        wrmsr

        /*
         * Finally, turn on paging (CR0.PG = 1) to activate long mode.
         */
        movl    %cr0, %eax
        orl     $CR0_PG, %eax
        movl    %eax, %cr0

        /*
         * The instruction after enabling paging in CR0 MUST be a branch.
         */
        jmp     long_mode_active

long_mode_active:
        /*
         * Long mode is now active but since we're still running with the
         * original 16-bit CS we're actually in 16-bit compatability mode.
         *
         * We have to load an intermediate GDT and IDT here that we know are
```



```
         * in 32-bit space before we can use the kernel's GDT and IDT, which
         * may be in the 64-bit address space, and since we're in compatability
         * mode, we only have access to 16 and 32-bit instructions at the
         * moment.
         */
        addr32 lgdtl    TEMPGDTOFF      /* load temporary GDT */
        addr32 lidtl    TEMPIDTOFF      /* load temporary IDT */

        /*
         * Do a far transfer to 64-bit mode.  Set the CS selector to a 64-bit
         * long mode selector (CS.L=1) in the temporary 32-bit GDT and jump
         * to the real mode platter address of long_mode 64 as until the 64-bit
         * CS is in place we don't have access to 64-bit instructions and thus
         * can't reference a 64-bit %rip.
         */
        pushl           $TEMP_CS64_SEL
        addr32 pushl    LM64OFF
        lretl

        .globl  long_mode_64
long_mode_64:
        .code64
        /*
         * We are now running in long mode with a 64-bit CS (EFER.LMA=1,
         * CS.L=1) so we now have access to 64-bit instructions.
         *
         * First, set the 64-bit GDT base.
         */
        .globl  rm_platter_pa
        movl    rm_platter_pa, %eax
        lgdtq   GDTROFF(%rax)           /* load 64-bit GDT */

        /*
         * Save the CPU number in %r11; get the value here since it's saved in
         * the real mode platter.
         */
        movl    CPUNOFF(%rax), %r11d

        /*
         * Add rm_platter_pa to %rsp to point it to the same location as seen
         * from 64-bit mode.
         */
        addq    %rax, %rsp

        /*
         * Now do an lretq to load CS with the appropriate selector for the
         * kernel's 64-bit GDT and to start executing 64-bit setup code at the
         * virtual address where boot originally loaded this code rather than
         * the copy in the real mode platter's rm_code array as we've been
         * doing so far.
         */
        pushq   $KCS_SEL
        pushq   $kernel_cs_code
        lretq
        .globl real_mode_start_cpu_end
real_mode_start_cpu_end:
        nop

kernel_cs_code:
        /*
         * Complete the balance of the setup we need to before executing
         * 64-bit kernel code (namely init rsp, TSS, LGDT, FS and GS).
         */
        .globl  rm_platter_va
        movq    rm_platter_va, %rax
        lidtq   IDTROFF(%rax)
```

```
259         movw    $KDS_SEL, %ax
260         movw    %ax, %ds
261         movw    %ax, %es
262         movw    %ax, %ss

264         movw    $KTSS_SEL, %ax          /* setup kernel TSS */
265         ltr     %ax

267         xorw    %ax, %ax                /* clear LDTR */
268         lldt    %ax

270         /*
271          * Set GS to the address of the per-cpu structure as contained in
272          * cpu[cpu_number].
273          *
274          * Unfortunately there's no way to set the 64-bit gsbase with a mov,
275          * so we have to stuff the low 32 bits in %eax and the high 32 bits in
276          * %edx, then call wrmsr.
277          */
278         leaq    cpu(%rip), %rdi
279         movl    (%rdi, %r11, 8), %eax
280         movl    4(%rdi, %r11, 8), %edx
281         movl    $MSR_AMD_GSBASE, %ecx
282         wrmsr

284         /*
285          * Init FS and KernelGSBase.
286          *
287          * Based on code in mlsetup(), set them both to 8G (which shouldn't be
288          * valid until some 64-bit processes run); this will then cause an
289          * exception in any code that tries to index off them before they are
290          * properly setup.
291          */
292         xorl    %eax, %eax              /* low 32 bits = 0 */
293         movl    $2, %edx                /* high 32 bits = 2 */
294         movl    $MSR_AMD_FSBASE, %ecx
295         wrmsr

297         movl    $MSR_AMD_KGSBASE, %ecx
298         wrmsr

300         /*
301          * Init %rsp to the exception stack set in tss_ist1 and create a legal
302          * AMD64 ABI stack frame
303          */
304         movq    %gs:CPU_TSS, %rax
305         movq    TSS_IST1(%rax), %rsp
306         pushq   $0              /* null return address */
307         pushq   $0              /* null frame pointer terminates stack trace */
308         movq    %rsp, %rbp      /* stack aligned on 16-byte boundary */

310         movq    %cr0, %rax
311         andq    $~(CR0_TS|CR0_EM), %rax /* clear emulate math chip bit */
312         orq     $(CR0_MP|CR0_NE), %rax
313         movq    %rax, %cr0              /* set machine status word */

315         /*
316          * Before going any further, enable usage of page table NX bit if
317          * that's how our page tables are set up.
318          */
319         bt      $X86FSET_NX, x86_featureset(%rip)
320         jnc     1f
321         movl    $MSR_AMD_EFER, %ecx
322         rdmsr
323         orl     $AMD_EFER_NXE, %eax
```



```
324         wrmsr
325 1:

327         /*
328          * Complete the rest of the setup and call mp_startup().
329          */
330         movq    %gs:CPU_THREAD, %rax    /* get thread ptr */
331         call    *T_PC(%rax)             /* call mp_startup_boot */
329         call    *T_PC(%rax)             /* call mp_startup */
332         /* not reached */
333         int     $20                     /* whoops, returned somehow! */

335         SET_SIZE(real_mode_start_cpu)

337 #elif defined(__i386)

339         ENTRY_NP(real_mode_start_cpu)

341 #if !defined(__GNUC_AS__)

343         cli
344         D16 movw        %cs, %eax
345         movw            %eax, %ds       /* load cs into ds */
346         movw            %eax, %ss       /* and into ss */

348         /*
349          * Helps in debugging by giving us the fault address.
350          *
351          * Remember to patch a hlt (0xf4) at cmntrap to get a good stack.
352          */
353         D16 movl        $0xffc, %esp

355         D16 A16 lgdt    %cs:GDTROFF
356         D16 A16 lidt    %cs:IDTROFF
357         D16 A16 movl    %cs:CR4OFF, %eax        /* set up CR4, if desired */
358         D16 andl        %eax, %eax
359         D16 A16 je      no_cr4

361         D16 movl        %eax, %ecx
362         D16 movl        %cr4, %eax
363         D16 orl         %ecx, %eax
364         D16 movl        %eax, %cr4
365 no_cr4:
366         D16 A16 movl    %cs:CR3OFF, %eax
367         A16 movl        %eax, %cr3
368         movl            %cr0, %eax

370         /*
371          * Enable protected-mode, paging, write protect, and alignment mask
372          */
373         D16 orl         $[CR0_PG|CR0_PE|CR0_WP|CR0_AM], %eax
374         movl            %eax, %cr0
375         jmp             pestart

377 pestart:
378         D16 pushl       $KCS_SEL
379         D16 pushl       $kernel_cs_code
380         D16 lret
381         .globl real_mode_start_cpu_end
382 real_mode_start_cpu_end:
383         nop

385         .globl  kernel_cs_code
386 kernel_cs_code:
387         /*
388          * At this point we are with kernel's cs and proper eip.
```

```
389          *
390          * We will be executing not from the copy in real mode platter,
391          * but from the original code where boot loaded us.
392          *
393          * By this time GDT and IDT are loaded as is cr3.
394          */
395         movw    $KFS_SEL,%eax
396         movw    %eax,%fs
397         movw    $KGS_SEL,%eax
398         movw    %eax,%gs
399         movw    $KDS_SEL,%eax
400         movw    %eax,%ds
401         movw    %eax,%es
402         movl    %gs:CPU_TSS,%esi
403         movw    %eax,%ss
404         movl    TSS_ESP0(%esi),%esp
405         movw    $KTSS_SEL,%ax
406         ltr     %ax
407         xorw    %ax, %ax                /* clear LDTR */
408         lldt    %ax
409         movl    %cr0,%edx
410         andl    $-1![CR0_TS|CR0_EM],%edx  /* clear emulate math chip bit */
411         orl     $[CR0_MP|CR0_NE],%edx
412         movl    %edx,%cr0                 /* set machine status word */

414         /*
415          * Before going any further, enable usage of page table NX bit if
416          * that's how our page tables are set up.
417          */
418         bt      $X86FSET_NX, x86_featureset
419         jnc     1f
420         movl    %cr4, %ecx
421         andl    $CR4_PAE, %ecx
422         jz      1f
423         movl    $MSR_AMD_EFER, %ecx
424         rdmsr
425         orl     $AMD_EFER_NXE, %eax
426         wrmsr
427 1:
428         movl    %gs:CPU_THREAD, %eax    /* get thread ptr */
429         call    *T_PC(%eax)             /* call mp_startup */
430         /* not reached */
431         int     $20                     /* whoops, returned somehow! */

433 #else

435         cli
436         mov             %cs, %ax
437         mov             %eax, %ds       /* load cs into ds */
438         mov             %eax, %ss       /* and into ss */

440         /*
441          * Helps in debugging by giving us the fault address.
442          *
443          * Remember to patch a hlt (0xf4) at cmntrap to get a good stack.
444          */
445         D16 mov         $0xffc, %esp

447         D16 A16 lgdtl   %cs:GDTROFF
448         D16 A16 lidtl   %cs:IDTROFF
449         D16 A16 mov     %cs:CR4OFF, %eax        /* set up CR4, if desired */
450         D16 and         %eax, %eax
451         D16 A16 je      no_cr4

453         D16 mov         %eax, %ecx
454         D16 mov         %cr4, %eax
```



```
455         D16 or          %ecx, %eax
456         D16 mov         %eax, %cr4
457 no_cr4:
458         D16 A16 mov     %cs:CR3OFF, %eax
459         A16 mov         %eax, %cr3
460         mov             %cr0, %eax

462         /*
463          * Enable protected-mode, paging, write protect, and alignment mask
464          */
465         D16 or          $(CR0_PG|CR0_PE|CR0_WP|CR0_AM), %eax
466         mov             %eax, %cr0
467         jmp             pestart

469 pestart:
470         D16 pushl       $KCS_SEL
471         D16 pushl       $kernel_cs_code
472         D16 lret
473         .globl real_mode_start_cpu_end
474 real_mode_start_cpu_end:
475         nop
476         .globl  kernel_cs_code
477 kernel_cs_code:
478         /*
479          * At this point we are with kernel's cs and proper eip.
480          *
481          * We will be executing not from the copy in real mode platter,
482          * but from the original code where boot loaded us.
483          *
484          * By this time GDT and IDT are loaded as is cr3.
485          */
486         mov     $KFS_SEL, %ax
487         mov     %eax, %fs
488         mov     $KGS_SEL, %ax
489         mov     %eax, %gs
490         mov     $KDS_SEL, %ax
491         mov     %eax, %ds
492         mov     %eax, %es
493         mov     %gs:CPU_TSS, %esi
494         mov     %eax, %ss
495         mov     TSS_ESP0(%esi), %esp
496         mov     $(KTSS_SEL), %ax
497         ltr     %ax
498         xorw    %ax, %ax                /* clear LDTR */
499         lldt    %ax
500         mov     %cr0, %edx
501         and     $~(CR0_TS|CR0_EM), %edx /* clear emulate math chip bit */
502         or      $(CR0_MP|CR0_NE), %edx
503         mov     %edx, %cr0              /* set machine status word */

505         /*
506          * Before going any farther, enable usage of page table NX bit if
507          * that's how our page tables are set up.  (PCIDE is enabled later on).
505          * that's how our page tables are set up.
508          */
509         bt      $X86FSET_NX, x86_featureset
510         jnc     1f
511         movl    %cr4, %ecx
512         andl    $CR4_PAE, %ecx
513         jz      1f
514         movl    $MSR_AMD_EFER, %ecx
515         rdmsr
516         orl     $AMD_EFER_NXE, %eax
517         wrmsr
518 1:
519         mov     %gs:CPU_THREAD, %eax    /* get thread ptr */
```

```
 520         call    *T_PC(%eax)             /* call mp_startup */
 521         /* not reached */
 522         int     $20                     /* whoops, returned somehow! */
 523 #endif

 525         SET_SIZE(real_mode_start_cpu)
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    8676 Fri Apr  6 17:25:00 2018
new/usr/src/uts/i86pc/ml/offsets.in
9441 kmdb should stash %cr3 in kdiregs
Reviewed by: John Levon <john.levon@joyent.com>
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
```

```
\
\ Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
\ Copyright 2012 Garrett D'Amore <garrett@damore.org>.  All rights reserved.
\ Copyright 2018 Joyent, Inc.
\ Copyright 2016 Joyent, Inc.
\
\ 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
\


\
\ offsets.in: input file to produce assym.h using the ctfstabs program
\

#ifndef _GENASSYM
#define _GENASSYM
#endif

#define SIZES   1

#include <sys/types.h>
#include <sys/bootsvcs.h>
#include <sys/systm.h>
#include <sys/sysinfo.h>
#include <sys/user.h>
#include <sys/thread.h>
#include <sys/proc.h>
#include <sys/cpuvar.h>
#include <sys/tss.h>
#include <sys/privregs.h>
#include <sys/segments.h>
#include <sys/devops.h>
#include <sys/ddi_impldefs.h>
#include <vm/as.h>
#include <sys/avintr.h>
#include <sys/pic.h>
#include <sys/rm_platter.h>
#include <sys/stream.h>
#include <sys/strsubr.h>
#include <sys/sunddi.h>
```



```
#include <sys/modctl.h>
#include <sys/traptrace.h>
#include <sys/ontrap.h>
#include <sys/lgrp.h>
#include <sys/dtrace.h>
#include <sys/brand.h>
#include <sys/fastboot.h>
#include <sys/cpr_wakecode.h>
#include <sys/comm_page.h>

proc            PROCSIZE
        p_link
        p_next
        p_child
        p_sibling
        p_sig
        p_flag
        p_tlist
        p_as
        p_lockp
        p_user
        p_ldt
        p_ldt_desc
        p_model
        p_pctx
        p_agenttp
        p_zone
        p_brand
        p_brand_data

_kthread        THREAD_SIZE
        t_pcb                   T_LABEL
        t_lock
        t_lockstat
        t_lockp
        t_lock_flush
        t_kpri_req
        t_oldspl
        t_pri
        t_pil
        t_lwp
        t_procp
        t_link
        t_state
        t_mstate
        t_preempt_lk
        t_stk                   T_STACK
        t_swap
        t_lwpchan.lc_wchan      T_WCHAN
        t_flag                  T_FLAGS
        t_ctx
        t_lofault
        t_onfault
        t_ontrap
        t_cpu
        t_lpl
        t_bound_cpu
        t_intr
        t_forw
        t_back
        t_sig
        t_tid
        t_pre_sys
        t_preempt
        t_proc_flag
        t_startpc
```

```
123         t_sysnum
124         t_intr_start
125         _tu._ts._t_astflag      T_ASTFLAG
126         _tu._ts._t_post_sys     T_POST_SYS
127         _tu._t_post_sys_ast     T_POST_SYS_AST
128         t_copyops
129 #ifdef  __amd64
130         t_useracc
131 #endif

133 ctxop
134         save_op                 CTXOP_SAVE

136 as
137         a_hat

139 user    USIZEBYTES
140         u_comm
141         u_signal

143 _label_t
144         val     LABEL_VAL

146 \#define        LABEL_PC        LABEL_VAL
147 \#define        LABEL_SP        _CONST(LABEL_VAL + LABEL_VAL_INCR)
148 \#define        T_PC            _CONST(T_LABEL + LABEL_PC)
149 \#define        T_SP            _CONST(T_LABEL + LABEL_SP)

151 _klwp
152         lwp_thread
153         lwp_procp
154         lwp_brand
155         lwp_eosys
156         lwp_regs
157         lwp_arg
158         lwp_ap
159         lwp_cursig
160         lwp_state
161         lwp_mstate.ms_acct      LWP_MS_ACCT
162         lwp_mstate.ms_prev      LWP_MS_PREV
163         lwp_mstate.ms_start     LWP_MS_START
164         lwp_mstate.ms_state_start LWP_MS_STATE_START
165         lwp_pcb
166         lwp_ru.sysc             LWP_RU_SYSC

168 \#define        LWP_ACCT_USER   _CONST(LWP_MS_ACCT + _MUL(LMS_USER, LWP_MS_ACCT_
169 \#define        LWP_ACCT_SYSTEM _CONST(LWP_MS_ACCT + _MUL(LMS_SYSTEM, LWP_MS_ACC

171 fpu_ctx
172         fpu_regs                FPU_CTX_FPU_REGS
173         fpu_flags               FPU_CTX_FPU_FLAGS
174         fpu_xsave_mask          FPU_CTX_FPU_XSAVE_MASK

176 fxsave_state    FXSAVE_STATE_SIZE
177         fx_fsw                  FXSAVE_STATE_FSW
178         fx_mxcsr_mask           FXSAVE_STATE_MXCSR_MASK


181 autovec         AUTOVECSIZE
182         av_vector
183         av_intarg1
184         av_intarg2
185         av_ticksp
186         av_link
187         av_prilevel
188         av_dip
```



```
190 av_head
191         avh_link
192         avh_hi_pri
193         avh_lo_pri

195 cpu
196         cpu_id
197         cpu_flags
198         cpu_self
199         cpu_thread
200         cpu_thread_lock
201         cpu_kprunrun
202         cpu_lwp
203         cpu_fpowner
204         cpu_idle_thread
205         cpu_intr_thread
206         cpu_intr_actv
207         cpu_base_spl
208         cpu_intr_stack
209         cpu_stats.sys.cpumigrate        CPU_STATS_SYS_CPUMIGRATE
210         cpu_stats.sys.intr              CPU_STATS_SYS_INTR
211         cpu_stats.sys.intrblk           CPU_STATS_SYS_INTRBLK
212         cpu_stats.sys.syscall           CPU_STATS_SYS_SYSCALL
213         cpu_profile_pc
214         cpu_profile_upc
215         cpu_profile_pil
216         cpu_ftrace.ftd_state            CPU_FTRACE_STATE
217         cpu_mstate
218         cpu_intracct

220 \#define        CPU_INTR_ACTV_REF       _CONST(CPU_INTR_ACTV + 2)

222 cpu
223         cpu_m.pil_high_start    CPU_PIL_HIGH_START
224         cpu_m.intrstat          CPU_INTRSTAT
225         cpu_m.mcpu_current_hat  CPU_CURRENT_HAT
226         cpu_m.mcpu_gdt          CPU_GDT
227         cpu_m.mcpu_idt          CPU_IDT
228         cpu_m.mcpu_tss          CPU_TSS
229         cpu_m.mcpu_softinfo     CPU_SOFTINFO
230         cpu_m.mcpu_pri          CPU_PRI
231 #if defined(__xpv)
232         cpu_m.mcpu_vcpu_info    CPU_VCPU_INFO
233 #endif

235 cpu
236         cpu_m.mcpu_kpti.kf_kernel_cr3   CPU_KPTI_KCR3
237         cpu_m.mcpu_kpti.kf_user_cr3     CPU_KPTI_UCR3
238         cpu_m.mcpu_kpti.kf_tr_rsp       CPU_KPTI_TR_RSP
239         cpu_m.mcpu_kpti.kf_tr_cr3       CPU_KPTI_TR_CR3
240         cpu_m.mcpu_kpti.kf_r13          CPU_KPTI_R13
241         cpu_m.mcpu_kpti.kf_r14          CPU_KPTI_R14
242         cpu_m.mcpu_kpti.kf_tr_ret_rsp   CPU_KPTI_RET_RSP

244         cpu_m.mcpu_kpti.kf_ss           CPU_KPTI_SS
245         cpu_m.mcpu_kpti.kf_rsp          CPU_KPTI_RSP
246         cpu_m.mcpu_kpti.kf_rflags       CPU_KPTI_RFLAGS
247         cpu_m.mcpu_kpti.kf_cs           CPU_KPTI_CS
248         cpu_m.mcpu_kpti.kf_rip          CPU_KPTI_RIP
249         cpu_m.mcpu_kpti.kf_err          CPU_KPTI_ERR

251         cpu_m.mcpu_pad2                 CPU_KPTI_START
252         cpu_m.mcpu_pad3                 CPU_KPTI_END

254         cpu_m.mcpu_kpti_dbg             CPU_KPTI_DBG
```

```
256 kpti_frame
257         kf_r14          KPTI_R14
258         kf_r13          KPTI_R13
259         kf_err          KPTI_ERR
260         kf_rip          KPTI_RIP
261         kf_cs           KPTI_CS
262         kf_rflags       KPTI_RFLAGS
263         kf_rsp          KPTI_RSP
264         kf_ss           KPTI_SS

266         kf_tr_rsp       KPTI_TOP

268         kf_kernel_cr3   KPTI_KCR3
269         kf_user_cr3     KPTI_UCR3
270         kf_tr_ret_rsp   KPTI_RET_RSP
271         kf_tr_cr3       KPTI_TR_CR3

273         kf_tr_flag      KPTI_FLAG

275 standard_pic
276         c_curmask
277         c_iplmask

279 ddi_dma_impl
280         dmai_rflags
281         dmai_rdip

283 dev_info
284         devi_ops                DEVI_DEV_OPS
285         devi_bus_ctl
286         devi_bus_dma_ctl
287         devi_bus_dma_allochdl
288         devi_bus_dma_freehdl
289         devi_bus_dma_bindhdl
290         devi_bus_dma_unbindhdl
291         devi_bus_dma_flush
292         devi_bus_dma_win

294 dev_ops
295         devo_bus_ops            DEVI_BUS_OPS

297 bus_ops
298         bus_ctl                 OPS_CTL
299         bus_dma_map             OPS_MAP
300         bus_dma_ctl             OPS_MCTL
301         bus_dma_allochdl        OPS_ALLOCHDL
302         bus_dma_freehdl         OPS_FREEHDL
303         bus_dma_bindhdl         OPS_BINDHDL
304         bus_dma_unbindhdl       OPS_UNBINDHDL
305         bus_dma_flush           OPS_FLUSH
306         bus_dma_win             OPS_WIN

308 sysent  SYSENT_SIZE     SYSENT_SIZE_SHIFT
309         sy_callc
310         sy_flags
311         sy_narg

313 stdata
314         sd_lock

316 queue
317         q_flag
318         q_next
319         q_stream
320         q_syncq
```



```
321         q_qinfo

323 qinit
324         qi_putp

326 syncq
327         sq_flags
328         sq_count
329         sq_lock
330         sq_wait

332 rm_platter
333         rm_idt_lim              IDTROFF
334         rm_gdt_lim              GDTROFF
335         rm_pdbr                 CR3OFF
336         rm_cpu                  CPUNOFF
337         rm_cr4                  CR4OFF
338         rm_cpu_halt_code        CPUHALTCODEOFF
339         rm_cpu_halted           CPUHALTEDOFF

341 ddi_acc_impl
342         ahi_acc_attr    ACC_ATTR
343         ahi_get8        ACC_GETB
344         ahi_get16       ACC_GETW
345         ahi_get32       ACC_GETL
346         ahi_get64       ACC_GETLL
347         ahi_put8        ACC_PUTB
348         ahi_put16       ACC_PUTW
349         ahi_put32       ACC_PUTL
350         ahi_put64       ACC_PUTLL
351         ahi_rep_get8    ACC_REP_GETB
352         ahi_rep_get16   ACC_REP_GETW
353         ahi_rep_get32   ACC_REP_GETL
354         ahi_rep_get64   ACC_REP_GETLL
355         ahi_rep_put8    ACC_REP_PUTB
356         ahi_rep_put16   ACC_REP_PUTW
357         ahi_rep_put32   ACC_REP_PUTL
358         ahi_rep_put64   ACC_REP_PUTLL

360 on_trap_data
361         ot_prot
362         ot_trap
363         ot_trampoline
364         ot_jmpbuf
365         ot_prev
366         ot_handle
367         ot_pad1

369 trap_trace_ctl_t         __TRAPTR_SIZE TRAPTR_SIZE_SHIFT
370         ttc_next        TRAPTR_NEXT
371         ttc_first       TRAPTR_FIRST
372         ttc_limit       TRAPTR_LIMIT

374 trap_trace_rec_t         TRAP_ENT_SIZE
375         ttr_cr2
376         ttr_info.idt_entry.vector       TTR_VECTOR
377         ttr_info.idt_entry.ipl          TTR_IPL
378         ttr_info.idt_entry.spl          TTR_SPL
379         ttr_info.idt_entry.pri          TTR_PRI
380         ttr_info.gate_entry.sysnum      TTR_SYSNUM
381         ttr_marker
382         ttr_stamp
383         ttr_curthread
384         ttr_sdepth
385         ttr_stack
```

```
387 lgrp_ld
388         lpl_lgrpid

390 dtrace_id_t     DTRACE_IDSIZE

392 cpu_core        CPU_CORE_SIZE   CPU_CORE_SHIFT
393         cpuc_dtrace_flags
394         cpuc_dtrace_illval

396 timespec        TIMESPEC_SIZE

398 gate_desc       GATE_DESC_SIZE

400 desctbr_t       DESCTBR_SIZE
401         dtr_limit
402         dtr_base

404 mod_stub_info   MODS_SIZE
405         mods_func_adr   MODS_INSTFCN
406         mods_errfcn     MODS_RETFCN
407         mods_flag

409 \#define        TRAP_TSIZE              _MUL(TRAP_ENT_SIZE, TRAPTR_NENT)

411 copyops
412         cp_copyin
413         cp_xcopyin
414         cp_copyout
415         cp_xcopyout
416         cp_copyinstr
417         cp_copyoutstr
418         cp_fuword8
419         cp_fuword16
420         cp_fuword32
421         cp_fuword64
422         cp_suword8
423         cp_suword16
424         cp_suword32
425         cp_suword64
426         cp_physio

428 brand
429         b_machops

431 brand_proc_data_t
432         spd_handler

434 fastboot_file_t
435         fb_va
436         fb_pte_list_va
437         fb_pte_list_pa
438         fb_dest_pa
439         fb_size
440         fb_next_pa
441         fb_sections
442         fb_sectcnt

444 fastboot_section_t
445         fb_sec_offset
446         fb_sec_paddr
447         fb_sec_size
448         fb_sec_bss_size

450 fastboot_info_t
451         fi_files
452         fi_has_pae
```



```
453         fi_pagetable_va
454         fi_pagetable_pa
455         fi_last_table_pa
456         fi_new_mbi_pa
457         fi_valid

459 zone
460         zone_brand_data

462 wc_cpu  WC_CPU_SIZE
463         wc_retaddr
464         wc_virtaddr
465         wc_cr0
466         wc_cr3
467         wc_cr4
468         wc_cr8
469         wc_fs
470         wc_fsbase
471         wc_gs
472         wc_gsbase
473         wc_kgsbase
474         wc_r8
475         wc_r9
476         wc_r10
477         wc_r11
478         wc_r12
479         wc_r13
480         wc_r14
481         wc_r15
482         wc_rax
483         wc_rbp
484         wc_rbx
485         wc_rcx
486         wc_rdi
487         wc_rdx
488         wc_rsi
489         wc_rsp
490         wc_gdt_limit    WC_GDT
491         wc_gdt_base
492         wc_idt_limit    WC_IDT
493         wc_idt_base
494         wc_tr
495         wc_ldt
496         wc_eflags
497         wc_ebx
498         wc_edi
499         wc_esi
500         wc_ebp
501         wc_esp
502         wc_esp
503         wc_ss
504         wc_cs
505         wc_ds
506         wc_es
507         wc_cpu_id
508         wc_saved_stack

510 wc_wakecode
511         wc_cpu

513 comm_page_s     COMM_PAGE_S_SIZE
```

```
**********************************************************
   38851 Fri Apr  6 17:25:01 2018
new/usr/src/uts/i86pc/ml/syscall_asm_amd64.s
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
  23  * Copyright 2018 Joyent, Inc.
  23  * Copyright 2015 Joyent, Inc.
  24  * Copyright (c) 2016 by Delphix. All rights reserved.
  25  */

  27 #include <sys/asm_linkage.h>
  28 #include <sys/asm_misc.h>
  29 #include <sys/regset.h>
  30 #include <sys/privregs.h>
  31 #include <sys/psw.h>
  32 #include <sys/machbrand.h>

  34 #if defined(__lint)

  36 #include <sys/types.h>
  37 #include <sys/thread.h>
  38 #include <sys/systm.h>

  40 #else   /* __lint */

  42 #include <sys/segments.h>
  43 #include <sys/pcb.h>
  44 #include <sys/trap.h>
  45 #include <sys/ftrace.h>
  46 #include <sys/traptrace.h>
  47 #include <sys/clock.h>
  48 #include <sys/model.h>
  49 #include <sys/panic.h>

  51 #if defined(__xpv)
  52 #include <sys/hypervisor.h>
  53 #endif

  55 #include "assym.h"

  57 #endif  /* __lint */
```



```
  59 /*
  60  * We implement five flavours of system call entry points
  61  *
  62  * -    syscall/sysretq         (amd64 generic)
  63  * -    syscall/sysretl         (i386 plus SYSC bit)
  64  * -    sysenter/sysexit        (i386 plus SEP bit)
  65  * -    int/iret                (i386 generic)
  66  * -    lcall/iret              (i386 generic)
  67  *
  68  * The current libc included in Solaris uses int/iret as the base unoptimized
  69  * kernel entry method. Older libc implementations and legacy binaries may use
  70  * the lcall call gate, so it must continue to be supported.
  71  *
  72  * System calls that use an lcall call gate are processed in trap() via a
  73  * segment-not-present trap, i.e. lcalls are extremely slow(!).
  74  *
  75  * The basic pattern used in the 32-bit SYSC handler at this point in time is
  76  * to have the bare minimum of assembler, and get to the C handlers as
  77  * quickly as possible.
  78  *
  79  * The 64-bit handler is much closer to the sparcv9 handler; that's
  80  * because of passing arguments in registers.  The 32-bit world still
  81  * passes arguments on the stack -- that makes that handler substantially
  82  * more complex.
  83  *
  84  * The two handlers share a few code fragments which are broken
  85  * out into preprocessor macros below.
  86  *
  87  * XX64 come back and speed all this up later.  The 32-bit stuff looks
  88  * especially easy to speed up the argument copying part ..
  89  *
  90  *
  91  * Notes about segment register usage (c.f. the 32-bit kernel)
  92  *
  93  * In the 32-bit kernel, segment registers are dutifully saved and
  94  * restored on all mode transitions because the kernel uses them directly.
  95  * When the processor is running in 64-bit mode, segment registers are
  96  * largely ignored.
  97  *
  98  * %cs and %ss
  99  *      controlled by the hardware mechanisms that make mode transitions
 100  *
 101  * The remaining segment registers have to either be pointing at a valid
 102  * descriptor i.e. with the 'present' bit set, or they can NULL descriptors
 103  *
 104  * %ds and %es
 105  *      always ignored
 106  *
 107  * %fs and %gs
 108  *      fsbase and gsbase are used to control the place they really point at.
 109  *      The kernel only depends on %gs, and controls its own gsbase via swapgs
 110  *
 111  * Note that loading segment registers is still costly because the GDT
 112  * lookup still happens (this is because the hardware can't know that we're
 113  * not setting up these segment registers for a 32-bit program).  Thus we
 114  * avoid doing this in the syscall path, and defer them to lwp context switch
 115  * handlers, so the register values remain virtualized to the lwp.
 116  */

 118 #if defined(SYSCALLTRACE)
 119 #define ORL_SYSCALLTRACE(r32)           \
 120         orl     syscalltrace(%rip), r32
 121 #else
 122 #define ORL_SYSCALLTRACE(r32)
 123 #endif
```

```
/*
 * In the 32-bit kernel, we do absolutely nothing before getting into the
 * brand callback checks.  In 64-bit land, we do swapgs and then come here.
 * We assume that the %rsp- and %r15-stashing fields in the CPU structure
 * are still unused.
 *
 * Check if a brand_mach_ops callback is defined for the specified callback_id
 * type.  If so invoke it with the kernel's %gs value loaded and the following
 * data on the stack:
 *
 * stack:  --------------------------------------
 *      32 | callback pointer                   |
 *    | 24 | user (or interrupt) stack pointer  |
 *    | 16 | lwp pointer                        |
 *    v  8 | userland return address            |
 *       0 | callback wrapper return addr       |
 *         --------------------------------------
 *
 * Since we're pushing the userland return address onto the kernel stack
 * we need to get that address without accessing the user's stack (since we
 * can't trust that data).  There are different ways to get the userland
 * return address depending on how the syscall trap was made:
 *
 * a) For sys_syscall and sys_syscall32 the return address is in %rcx.
 * b) For sys_sysenter the return address is in %rdx.
 * c) For sys_int80 and sys_syscall_int (int91), upon entry into the macro,
 *    the stack pointer points at the state saved when we took the interrupt:
 *       ------------------------
 *    |  | user's %ss           |
 *    |  | user's %esp          |
 *    |  | EFLAGS register      |
 *    v  | user's %cs           |
 *       | user's %eip          |
 *       ------------------------
 *
 * The 2nd parameter to the BRAND_CALLBACK macro is either the
 * BRAND_URET_FROM_REG or BRAND_URET_FROM_INTR_STACK macro.  These macros are
 * used to generate the proper code to get the userland return address for
 * each syscall entry point.
 *
 * The interface to the brand callbacks on the 64-bit kernel assumes %r15
 * is available as a scratch register within the callback.  If the callback
 * returns within the kernel then this macro will restore %r15.  If the
 * callback is going to return directly to userland then it should restore
 * %r15 before returning to userland.
 */
#define BRAND_URET_FROM_REG(rip_reg)                                    \
        pushq   rip_reg                 /* push the return address      */

/*
 * The interrupt stack pointer we saved on entry to the BRAND_CALLBACK macro
 * is currently pointing at the user return address (%eip).
 */
#define BRAND_URET_FROM_INTR_STACK()                                    \
        movq    %gs:CPU_RTMP_RSP, %r15  /* grab the intr. stack pointer */ ;\
        pushq   (%r15)                  /* push the return address      */

#define BRAND_CALLBACK(callback_id, push_userland_ret)                      \
        movq    %rsp, %gs:CPU_RTMP_RSP  /* save the stack pointer       */ ;\
        movq    %r15, %gs:CPU_RTMP_R15  /* save %r15                    */ ;\
        movq    %gs:CPU_THREAD, %r15    /* load the thread pointer      */ ;\
        movq    T_STACK(%r15), %rsp     /* switch to the kernel stack   */ ;\
        subq    $16, %rsp               /* save space for 2 pointers    */ ;\
        pushq   %r14                    /* save %r14                    */ ;\
        movq    %gs:CPU_RTMP_RSP, %r14                                      ;\
        movq    %r14, 8(%rsp)           /* stash the user stack pointer */ ;\
```



```
        popq    %r14                    /* restore %r14                 */ ;\
        movq    T_LWP(%r15), %r15       /* load the lwp pointer         */ ;\
        pushq   %r15                    /* push the lwp pointer         */ ;\
        movq    LWP_PROCP(%r15), %r15   /* load the proc pointer        */ ;\
        movq    P_BRAND(%r15), %r15     /* load the brand pointer       */ ;\
        movq    B_MACHOPS(%r15), %r15   /* load the machops pointer     */ ;\
        movq    _CONST(_MUL(callback_id, CPTRSIZE))(%r15), %r15             ;\
        cmpq    $0, %r15                                                    ;\
        je      1f                                                          ;\
        movq    %r15, 16(%rsp)          /* save the callback pointer    */ ;\
        push_userland_ret               /* push the return address      */ ;\
        call    *24(%rsp)               /* call callback                */ ;\
1:      movq    %gs:CPU_RTMP_R15, %r15  /* restore %r15                 */ ;\
        movq    %gs:CPU_RTMP_RSP, %rsp  /* restore the stack pointer    */

#define MSTATE_TRANSITION(from, to)             \
        movl    $from, %edi;                    \
        movl    $to, %esi;                      \
        call    syscall_mstate

/*
 * Check to see if a simple (direct) return is possible i.e.
 *
 *      if (t->t_post_sys_ast | syscalltrace |
 *          lwp->lwp_pcb.pcb_rupdate == 1)
 *              do full version ;
 *
 * Preconditions:
 * -    t is curthread
 * Postconditions:
 * -    condition code NE is set if post-sys is too complex
 * -    rtmp is zeroed if it isn't (we rely on this!)
 * -    ltmp is smashed
 */
#define CHECK_POSTSYS_NE(t, ltmp, rtmp)                 \
        movq    T_LWP(t), ltmp;                         \
        movzbl  PCB_RUPDATE(ltmp), rtmp;                \
        ORL_SYSCALLTRACE(rtmp);                         \
        orl     T_POST_SYS_AST(t), rtmp;                \
        cmpl    $0, rtmp

/*
 * Fix up the lwp, thread, and eflags for a successful return
 *
 * Preconditions:
 * -    zwreg contains zero
 */
#define SIMPLE_SYSCALL_POSTSYS(t, lwp, zwreg)           \
        movb    $LWP_USER, LWP_STATE(lwp);              \
        movw    zwreg, T_SYSNUM(t);                     \
        andb    $_CONST(0xffff - PS_C), REGOFF_RFL(%rsp)

/*
 * ASSERT(lwptoregs(lwp) == rp);
 *
 * This may seem obvious, but very odd things happen if this
 * assertion is false
 *
 * Preconditions:
 *      (%rsp is ready for normal call sequence)
 * Postconditions (if assertion is true):
 *      %r11 is smashed
 *
 * ASSERT(rp->r_cs == descnum)
 *
 * The code selector is written into the regs structure when the
```

```
 257  * lwp stack is created.  We use this ASSERT to validate that
 258  * the regs structure really matches how we came in.
 259  *
 260  * Preconditions:
 261  *      (%rsp is ready for normal call sequence)
 262  * Postconditions (if assertion is true):
 263  *      -none-
 264  *
 265  * ASSERT(lwp->lwp_pcb.pcb_rupdate == 0);
 266  *
 267  * If this is false, it meant that we returned to userland without
 268  * updating the segment registers as we were supposed to.
 269  *
 270  * Note that we must ensure no interrupts or other traps intervene
 271  * between entering privileged mode and performing the assertion,
 272  * otherwise we may perform a context switch on the thread, which
 273  * will end up setting pcb_rupdate to 1 again.
 274  */
 275 #if defined(DEBUG)

 277 #if !defined(__lint)

 279 __lwptoregs_msg:
 280         .string "syscall_asm_amd64.s:%d lwptoregs(%p) [%p] != rp [%p]"

 282 __codesel_msg:
 283         .string "syscall_asm_amd64.s:%d rp->r_cs [%ld] != %ld"

 285 __no_rupdate_msg:
 286         .string "syscall_asm_amd64.s:%d lwp %p, pcb_rupdate != 0"

 288 #endif  /* !__lint */

 290 #define ASSERT_LWPTOREGS(lwp, rp)                       \
 291         movq    LWP_REGS(lwp), %r11;                    \
 292         cmpq    rp, %r11;                               \
 293         je      7f;                                     \
 294         leaq    __lwptoregs_msg(%rip), %rdi;            \
 295         movl    $__LINE__, %esi;                        \
 296         movq    lwp, %rdx;                              \
 297         movq    %r11, %rcx;                             \
 298         movq    rp, %r8;                                \
 299         xorl    %eax, %eax;                             \
 300         call    panic;                                  \
 301 7:

 303 #define ASSERT_NO_RUPDATE_PENDING(lwp)                  \
 304         testb   $0x1, PCB_RUPDATE(lwp);                 \
 305         je      8f;                                     \
 306         movq    lwp, %rdx;                              \
 307         leaq    __no_rupdate_msg(%rip), %rdi;           \
 308         movl    $__LINE__, %esi;                        \
 309         xorl    %eax, %eax;                             \
 310         call    panic;                                  \
 311 8:

 313 #else
 314 #define ASSERT_LWPTOREGS(lwp, rp)
 315 #define ASSERT_NO_RUPDATE_PENDING(lwp)
 316 #endif

 318 /*
 319  * Do the traptrace thing and restore any registers we used
 320  * in situ.  Assumes that %rsp is pointing at the base of
 321  * the struct regs, obviously ..
 322  */
```



```
 323 #ifdef TRAPTRACE
 324 #define SYSCALL_TRAPTRACE(ttype)                                \
 325         TRACE_PTR(%rdi, %rbx, %ebx, %rcx, ttype);               \
 326         TRACE_REGS(%rdi, %rsp, %rbx, %rcx);                     \
 327         TRACE_STAMP(%rdi);      /* rdtsc clobbers %eax, %edx */ \
 328         movq    REGOFF_RAX(%rsp), %rax;                         \
 329         movq    REGOFF_RBX(%rsp), %rbx;                         \
 330         movq    REGOFF_RCX(%rsp), %rcx;                         \
 331         movq    REGOFF_RDX(%rsp), %rdx;                         \
 332         movl    %eax, TTR_SYSNUM(%rdi);                         \
 333         movq    REGOFF_RDI(%rsp), %rdi

 335 #define SYSCALL_TRAPTRACE32(ttype)                              \
 336         SYSCALL_TRAPTRACE(ttype);                               \
 337         /* paranoia: clean the top 32-bits of the registers */  \
 338         orl     %eax, %eax;                                     \
 339         orl     %ebx, %ebx;                                     \
 340         orl     %ecx, %ecx;                                     \
 341         orl     %edx, %edx;                                     \
 342         orl     %edi, %edi
 343 #else   /* TRAPTRACE */
 344 #define SYSCALL_TRAPTRACE(ttype)
 345 #define SYSCALL_TRAPTRACE32(ttype)
 346 #endif  /* TRAPTRACE */

 348 /*
 349  * The 64-bit libc syscall wrapper does this:
 350  *
 351  * fn(<args>)
 352  * {
 353  *      movq    %rcx, %r10      -- because syscall smashes %rcx
 354  *      movl    $CODE, %eax
 355  *      syscall
 356  *      <error processing>
 357  * }
 358  *
 359  * Thus when we come into the kernel:
 360  *
 361  *      %rdi, %rsi, %rdx, %r10, %r8, %r9 contain first six args
 362  *      %rax is the syscall number
 363  *      %r12-%r15 contain caller state
 364  *
 365  * The syscall instruction arranges that:
 366  *
 367  *      %rcx contains the return %rip
 368  *      %r11d contains bottom 32-bits of %rflags
 369  *      %rflags is masked (as determined by the SFMASK msr)
 370  *      %cs is set to UCS_SEL (as determined by the STAR msr)
 371  *      %ss is set to UDS_SEL (as determined by the STAR msr)
 372  *      %rip is set to sys_syscall (as determined by the LSTAR msr)
 373  *
 374  * Or in other words, we have no registers available at all.
 375  * Only swapgs can save us!
 376  *
 377  * Under the hypervisor, the swapgs has happened already.  However, the
 378  * state of the world is very different from that we're familiar with.
 379  *
 380  * In particular, we have a stack structure like that for interrupt
 381  * gates, except that the %cs and %ss registers are modified for reasons
 382  * that are not entirely clear.  Critically, the %rcx/%r11 values do
 383  * *not* reflect the usage of those registers under a 'real' syscall[1];
 384  * the stack, therefore, looks like this:
 385  *
 386  *      0x0(rsp)        potentially junk %rcx
 387  *      0x8(rsp)        potentially junk %r11
 388  *      0x10(rsp)       user %rip
```

```
 *      0x18(rsp)       modified %cs
 *      0x20(rsp)       user %rflags
 *      0x28(rsp)       user %rsp
 *      0x30(rsp)       modified %ss
 *
 *
 * and before continuing on, we must load the %rip into %rcx and the
 * %rflags into %r11.
 *
 * [1] They used to, and we relied on it, but this was broken in 3.1.1.
 * Sigh.
 */
#if defined(__xpv)
#define XPV_SYSCALL_PROD                                                \
        movq    0x10(%rsp), %rcx;                                       \
        movq    0x20(%rsp), %r11;                                       \
        movq    0x28(%rsp), %rsp
#else
#define XPV_SYSCALL_PROD /* nothing */
#endif

#if defined(__lint)

/*ARGSUSED*/
void
sys_syscall()
{}

void
_allsyscalls()
{}

size_t _allsyscalls_size;

#else   /* __lint */

        ENTRY_NP2(brand_sys_syscall,_allsyscalls)
        SWAPGS                          /* kernel gsbase */
        XPV_SYSCALL_PROD
        BRAND_CALLBACK(BRAND_CB_SYSCALL, BRAND_URET_FROM_REG(%rcx))
        jmp     noprod_sys_syscall

        ALTENTRY(sys_syscall)
        SWAPGS                          /* kernel gsbase */
        XPV_SYSCALL_PROD

noprod_sys_syscall:
        movq    %r15, %gs:CPU_RTMP_R15
        movq    %rsp, %gs:CPU_RTMP_RSP

        movq    %gs:CPU_THREAD, %r15
        movq    T_STACK(%r15), %rsp     /* switch from user to kernel stack */

        ASSERT_UPCALL_MASK_IS_SET

        movl    $UCS_SEL, REGOFF_CS(%rsp)
        movq    %rcx, REGOFF_RIP(%rsp)          /* syscall: %rip -> %rcx */
        movq    %r11, REGOFF_RFL(%rsp)          /* syscall: %rfl -> %r11d */
        movl    $UDS_SEL, REGOFF_SS(%rsp)

        movl    %eax, %eax                      /* wrapper: sysc# -> %eax */
        movq    %rdi, REGOFF_RDI(%rsp)
        movq    %rsi, REGOFF_RSI(%rsp)
        movq    %rdx, REGOFF_RDX(%rsp)
        movq    %r10, REGOFF_RCX(%rsp)          /* wrapper: %rcx -> %r10 */
        movq    %r10, %rcx                      /* arg[3] for direct calls */
```



```
        movq    %r8, REGOFF_R8(%rsp)
        movq    %r9, REGOFF_R9(%rsp)
        movq    %rax, REGOFF_RAX(%rsp)
        movq    %rbx, REGOFF_RBX(%rsp)

        movq    %rbp, REGOFF_RBP(%rsp)
        movq    %r10, REGOFF_R10(%rsp)
        movq    %gs:CPU_RTMP_RSP, %r11
        movq    %r11, REGOFF_RSP(%rsp)
        movq    %r12, REGOFF_R12(%rsp)

        movq    %r13, REGOFF_R13(%rsp)
        movq    %r14, REGOFF_R14(%rsp)
        movq    %gs:CPU_RTMP_R15, %r10
        movq    %r10, REGOFF_R15(%rsp)
        movq    $0, REGOFF_SAVFP(%rsp)
        movq    $0, REGOFF_SAVPC(%rsp)

        /*
         * Copy these registers here in case we end up stopped with
         * someone (like, say, /proc) messing with our register state.
         * We don't -restore- them unless we have to in update_sregs.
         *
         * Since userland -can't- change fsbase or gsbase directly,
         * and capturing them involves two serializing instructions,
         * we don't bother to capture them here.
         */
        xorl    %ebx, %ebx
        movw    %ds, %bx
        movq    %rbx, REGOFF_DS(%rsp)
        movw    %es, %bx
        movq    %rbx, REGOFF_ES(%rsp)
        movw    %fs, %bx
        movq    %rbx, REGOFF_FS(%rsp)
        movw    %gs, %bx
        movq    %rbx, REGOFF_GS(%rsp)

        /*
         * If we're trying to use TRAPTRACE though, I take that back: we're
         * probably debugging some problem in the SWAPGS logic and want to know
         * what the incoming gsbase was.
         *
         * Since we already did SWAPGS, record the KGSBASE.
         */
#if defined(DEBUG) && defined(TRAPTRACE) && !defined(__xpv)
        movl    $MSR_AMD_KGSBASE, %ecx
        rdmsr
        movl    %eax, REGOFF_GSBASE(%rsp)
        movl    %edx, REGOFF_GSBASE+4(%rsp)
#endif

        /*
         * Machine state saved in the regs structure on the stack
         * First six args in %rdi, %rsi, %rdx, %rcx, %r8, %r9
         * %eax is the syscall number
         * %rsp is the thread's stack, %r15 is curthread
         * REG_RSP(%rsp) is the user's stack
         */

        SYSCALL_TRAPTRACE($TT_SYSC64)

        movq    %rsp, %rbp

        movq    T_LWP(%r15), %r14
        ASSERT_NO_RUPDATE_PENDING(%r14)
```

```
521         ENABLE_INTR_FLAGS

523         MSTATE_TRANSITION(LMS_USER, LMS_SYSTEM)
524         movl    REGOFF_RAX(%rsp), %eax  /* (%rax damaged by mstate call) */

526         ASSERT_LWPTOREGS(%r14, %rsp)

528         movb    $LWP_SYS, LWP_STATE(%r14)
529         incq    LWP_RU_SYSC(%r14)
530         movb    $NORMALRETURN, LWP_EOSYS(%r14)

532         incq    %gs:CPU_STATS_SYS_SYSCALL

534         movw    %ax, T_SYSNUM(%r15)
535         movzbl  T_PRE_SYS(%r15), %ebx
536         ORL_SYSCALLTRACE(%ebx)
537         testl   %ebx, %ebx
538         jne     _syscall_pre

540 _syscall_invoke:
541         movq    REGOFF_RDI(%rbp), %rdi
542         movq    REGOFF_RSI(%rbp), %rsi
543         movq    REGOFF_RDX(%rbp), %rdx
544         movq    REGOFF_RCX(%rbp), %rcx
545         movq    REGOFF_R8(%rbp), %r8
546         movq    REGOFF_R9(%rbp), %r9

548         cmpl    $NSYSCALL, %eax
549         jae     _syscall_ill
550         shll    $SYSENT_SIZE_SHIFT, %eax
551         leaq    sysent(%rax), %rbx

553         call    *SY_CALLC(%rbx)

555         movq    %rax, %r12
556         movq    %rdx, %r13

558         /*
559          * If the handler returns two ints, then we need to split the
560          * 64-bit return value into two 32-bit values.
561          */
562         testw   $SE_32RVAL2, SY_FLAGS(%rbx)
563         je      5f
564         movq    %r12, %r13
565         shrq    $32, %r13       /* upper 32-bits into %edx */
566         movl    %r12d, %r12d    /* lower 32-bits into %eax */
567 5:
568         /*
569          * Optimistically assume that there's no post-syscall
570          * work to do.  (This is to avoid having to call syscall_mstate()
571          * with interrupts disabled)
572          */
573         MSTATE_TRANSITION(LMS_SYSTEM, LMS_USER)

575         /*
576          * We must protect ourselves from being descheduled here;
577          * If we were, and we ended up on another cpu, or another
578          * lwp got in ahead of us, it could change the segment
579          * registers without us noticing before we return to userland.
580          */
581         CLI(%r14)
582         CHECK_POSTSYS_NE(%r15, %r14, %ebx)
583         jne     _syscall_post

585         /*
586          * We need to protect ourselves against non-canonical return values
```



```
587          * because Intel doesn't check for them on sysret (AMD does).  Canonical
588          * addresses on current amd64 processors only use 48-bits for VAs; an
589          * address is canonical if all upper bits (47-63) are identical. If we
590          * find a non-canonical %rip, we opt to go through the full
591          * _syscall_post path which takes us into an iretq which is not
592          * susceptible to the same problems sysret is.
593          *
594          * We're checking for a canonical address by first doing an arithmetic
595          * shift. This will fill in the remaining bits with the value of bit 63.
596          * If the address were canonical, the register would now have either all
597          * zeroes or all ones in it. Therefore we add one (inducing overflow)
598          * and compare against 1. A canonical address will either be zero or one
599          * at this point, hence the use of ja.
600          *
601          * At this point, r12 and r13 have the return value so we can't use
602          * those registers.
603          */
604         movq    REGOFF_RIP(%rsp), %rcx
605         sarq    $47, %rcx
606         incq    %rcx
607         cmpq    $1, %rcx
608         ja      _syscall_post


611         SIMPLE_SYSCALL_POSTSYS(%r15, %r14, %bx)

613         movq    %r12, REGOFF_RAX(%rsp)
614         movq    %r13, REGOFF_RDX(%rsp)

616         /*
617          * To get back to userland, we need the return %rip in %rcx and
618          * the return %rfl in %r11d.  The sysretq instruction also arranges
619          * to fix up %cs and %ss; everything else is our responsibility.
620          */
621         movq    REGOFF_RDI(%rsp), %rdi
622         movq    REGOFF_RSI(%rsp), %rsi
623         movq    REGOFF_RDX(%rsp), %rdx
624         /* %rcx used to restore %rip value */

626         movq    REGOFF_R8(%rsp), %r8
627         movq    REGOFF_R9(%rsp), %r9
628         movq    REGOFF_RAX(%rsp), %rax
629         movq    REGOFF_RBX(%rsp), %rbx

631         movq    REGOFF_RBP(%rsp), %rbp
632         movq    REGOFF_R10(%rsp), %r10
633         /* %r11 used to restore %rfl value */
634         movq    REGOFF_R12(%rsp), %r12

636         movq    REGOFF_R13(%rsp), %r13
637         movq    REGOFF_R14(%rsp), %r14
638         movq    REGOFF_R15(%rsp), %r15

640         movq    REGOFF_RIP(%rsp), %rcx
641         movl    REGOFF_RFL(%rsp), %r11d

643 #if defined(__xpv)
644         addq    $REGOFF_RIP, %rsp
645 #else
646         movq    REGOFF_RSP(%rsp), %rsp
647 #endif

649         /*
650          * There can be no instructions between the ALTENTRY below and
651          * SYSRET or we could end up breaking brand support. See label usage
652          * in sn1_brand_syscall_callback for an example.
```

```
 653          */
 654         ASSERT_UPCALL_MASK_IS_SET
 655 #if defined(__xpv)
 656         SYSRETQ
 657         ALTENTRY(nopop_sys_syscall_swapgs_sysretq)

 659         /*
 660          * We can only get here after executing a brand syscall
 661          * interposition callback handler and simply need to
 662          * "sysretq" back to userland. On the hypervisor this
 663          * involves the iret hypercall which requires us to construct
 664          * just enough of the stack needed for the hypercall.
 665          * (rip, cs, rflags, rsp, ss).
 666          */
 667         movq    %rsp, %gs:CPU_RTMP_RSP          /* save user's rsp */
 668         movq    %gs:CPU_THREAD, %r11
 669         movq    T_STACK(%r11), %rsp

 671         movq    %rcx, REGOFF_RIP(%rsp)
 672         movl    $UCS_SEL, REGOFF_CS(%rsp)
 673         movq    %gs:CPU_RTMP_RSP, %r11
 674         movq    %r11, REGOFF_RSP(%rsp)
 675         pushfq
 676         popq    %r11                            /* hypercall enables ints */
 677         movq    %r11, REGOFF_RFL(%rsp)
 678         movl    $UDS_SEL, REGOFF_SS(%rsp)
 679         addq    $REGOFF_RIP, %rsp
 680         /*
 681          * XXPV: see comment in SYSRETQ definition for future optimization
 682          *       we could take.
 683          */
 684         ASSERT_UPCALL_MASK_IS_SET
 685         SYSRETQ
 686 #else
 687         ALTENTRY(nopop_sys_syscall_swapgs_sysretq)
 688         jmp     tr_sysretq
 674         SWAPGS                          /* user gsbase */
 675         SYSRETQ
 689 #endif
 690         /*NOTREACHED*/
 691         SET_SIZE(nopop_sys_syscall_swapgs_sysretq)
_____unchanged_portion_omitted_

 726 #endif  /* __lint */

 728 #if defined(__lint)

 730 /*ARGSUSED*/
 731 void
 732 sys_syscall32()
 733 {}

 735 #else   /* __lint */

 737         ENTRY_NP(brand_sys_syscall32)
 738         SWAPGS                          /* kernel gsbase */
 739         XPV_TRAP_POP
 740         BRAND_CALLBACK(BRAND_CB_SYSCALL32, BRAND_URET_FROM_REG(%rcx))
 741         jmp     nopop_sys_syscall32

 743         ALTENTRY(sys_syscall32)
 744         SWAPGS                          /* kernel gsbase */
 745         XPV_TRAP_POP

 747 nopop_sys_syscall32:
 748         movl    %esp, %r10d
```



```
 749         movq    %gs:CPU_THREAD, %r15
 750         movq    T_STACK(%r15), %rsp
 751         movl    %eax, %eax

 753         movl    $U32CS_SEL, REGOFF_CS(%rsp)
 754         movl    %ecx, REGOFF_RIP(%rsp)          /* syscall: %rip -> %rcx */
 755         movq    %r11, REGOFF_RFL(%rsp)          /* syscall: %rfl -> %r11d */
 756         movq    %r10, REGOFF_RSP(%rsp)
 757         movl    $UDS_SEL, REGOFF_SS(%rsp)

 759 _syscall32_save:
 760         movl    %edi, REGOFF_RDI(%rsp)
 761         movl    %esi, REGOFF_RSI(%rsp)
 762         movl    %ebp, REGOFF_RBP(%rsp)
 763         movl    %ebx, REGOFF_RBX(%rsp)
 764         movl    %edx, REGOFF_RDX(%rsp)
 765         movl    %ecx, REGOFF_RCX(%rsp)
 766         movl    %eax, REGOFF_RAX(%rsp)          /* wrapper: sysc# -> %eax */
 767         movq    $0, REGOFF_SAVFP(%rsp)
 768         movq    $0, REGOFF_SAVPC(%rsp)

 770         /*
 771          * Copy these registers here in case we end up stopped with
 772          * someone (like, say, /proc) messing with our register state.
 773          * We don't -restore- them unless we have to in update_sregs.
 774          *
 775          * Since userland -can't- change fsbase or gsbase directly,
 776          * we don't bother to capture them here.
 777          */
 778         xorl    %ebx, %ebx
 779         movw    %ds, %bx
 780         movq    %rbx, REGOFF_DS(%rsp)
 781         movw    %es, %bx
 782         movq    %rbx, REGOFF_ES(%rsp)
 783         movw    %fs, %bx
 784         movq    %rbx, REGOFF_FS(%rsp)
 785         movw    %gs, %bx
 786         movq    %rbx, REGOFF_GS(%rsp)

 788         /*
 789          * If we're trying to use TRAPTRACE though, I take that back: we're
 790          * probably debugging some problem in the SWAPGS logic and want to know
 791          * what the incoming gsbase was.
 792          *
 793          * Since we already did SWAPGS, record the KGSBASE.
 794          */
 795 #if defined(DEBUG) && defined(TRAPTRACE) && !defined(__xpv)
 796         movl    $MSR_AMD_KGSBASE, %ecx
 797         rdmsr
 798         movl    %eax, REGOFF_GSBASE(%rsp)
 799         movl    %edx, REGOFF_GSBASE+4(%rsp)
 800 #endif

 802         /*
 803          * Application state saved in the regs structure on the stack
 804          * %eax is the syscall number
 805          * %rsp is the thread's stack, %r15 is curthread
 806          * REG_RSP(%rsp) is the user's stack
 807          */

 809         SYSCALL_TRAPTRACE32($TT_SYSC)

 811         movq    %rsp, %rbp

 813         movq    T_LWP(%r15), %r14
 814         ASSERT_NO_RUPDATE_PENDING(%r14)
```

```
816         ENABLE_INTR_FLAGS

818         MSTATE_TRANSITION(LMS_USER, LMS_SYSTEM)
819         movl    REGOFF_RAX(%rsp), %eax  /* (%rax damaged by mstate call) */

821         ASSERT_LWPTOREGS(%r14, %rsp)

823         incq     %gs:CPU_STATS_SYS_SYSCALL

825         /*
826          * Make some space for MAXSYSARGS (currently 8) 32-bit args placed
827          * into 64-bit (long) arg slots, maintaining 16 byte alignment.  Or
828          * more succinctly:
829          *
830          *      SA(MAXSYSARGS * sizeof (long)) == 64
831          */
832 #define SYS_DROP        64                      /* drop for args */
833         subq    $SYS_DROP, %rsp
834         movb    $LWP_SYS, LWP_STATE(%r14)
835         movq    %r15, %rdi
836         movq    %rsp, %rsi
837         call    syscall_entry

839         /*
840          * Fetch the arguments copied onto the kernel stack and put
841          * them in the right registers to invoke a C-style syscall handler.
842          * %rax contains the handler address.
843          *
844          * Ideas for making all this go faster of course include simply
845          * forcibly fetching 6 arguments from the user stack under lofault
846          * protection, reverting to copyin_args only when watchpoints
847          * are in effect.
848          *
849          * (If we do this, make sure that exec and libthread leave
850          * enough space at the top of the stack to ensure that we'll
851          * never do a fetch from an invalid page.)
852          *
853          * Lots of ideas here, but they won't really help with bringup B-)
854          * Correctness can't wait, performance can wait a little longer ..
855          */

857         movq    %rax, %rbx
858         movl    0(%rsp), %edi
859         movl    8(%rsp), %esi
860         movl    0x10(%rsp), %edx
861         movl    0x18(%rsp), %ecx
862         movl    0x20(%rsp), %r8d
863         movl    0x28(%rsp), %r9d

865         call    *SY_CALLC(%rbx)

867         movq    %rbp, %rsp      /* pop the args */

869         /*
870          * amd64 syscall handlers -always- return a 64-bit value in %rax.
871          * On the 32-bit kernel, they always return that value in %eax:%edx
872          * as required by the 32-bit ABI.
873          *
874          * Simulate the same behaviour by unconditionally splitting the
875          * return value in the same way.
876          */
877         movq    %rax, %r13
878         shrq    $32, %r13       /* upper 32-bits into %edx */
879         movl    %eax, %r12d     /* lower 32-bits into %eax */
```



```
881         /*
882          * Optimistically assume that there's no post-syscall
883          * work to do.  (This is to avoid having to call syscall_mstate()
884          * with interrupts disabled)
885          */
886         MSTATE_TRANSITION(LMS_SYSTEM, LMS_USER)

888         /*
889          * We must protect ourselves from being descheduled here;
890          * If we were, and we ended up on another cpu, or another
891          * lwp got in ahead of us, it could change the segment
892          * registers without us noticing before we return to userland.
893          */
894         CLI(%r14)
895         CHECK_POSTSYS_NE(%r15, %r14, %ebx)
896         jne     _full_syscall_postsys32
897         SIMPLE_SYSCALL_POSTSYS(%r15, %r14, %bx)

899         /*
900          * To get back to userland, we need to put the return %rip in %rcx and
901          * the return %rfl in %r11d.  The sysret instruction also arranges
902          * to fix up %cs and %ss; everything else is our responsibility.
903          */

905         movl    %r12d, %eax                     /* %eax: rval1 */
906         movl    REGOFF_RBX(%rsp), %ebx
907         /* %ecx used for return pointer */
908         movl    %r13d, %edx                     /* %edx: rval2 */
909         movl    REGOFF_RBP(%rsp), %ebp
910         movl    REGOFF_RSI(%rsp), %esi
911         movl    REGOFF_RDI(%rsp), %edi

913         movl    REGOFF_RFL(%rsp), %r11d         /* %r11 -> eflags */
914         movl    REGOFF_RIP(%rsp), %ecx          /* %ecx -> %eip */
915         movl    REGOFF_RSP(%rsp), %esp

917         ASSERT_UPCALL_MASK_IS_SET
918         ALTENTRY(nopop_sys_syscall32_swapgs_sysretl)
919         jmp     tr_sysretl
892         SWAPGS                          /* user gsbase */
893         SYSRETL
920         SET_SIZE(nopop_sys_syscall32_swapgs_sysretl)
_____unchanged_portion_omitted_

939 #endif  /* __lint */

941 /*
942  * System call handler via the sysenter instruction
943  * Used only for 32-bit system calls on the 64-bit kernel.
944  *
945  * The caller in userland has arranged that:
946  *
947  * -    %eax contains the syscall number
948  * -    %ecx contains the user %esp
949  * -    %edx contains the return %eip
950  * -    the user stack contains the args to the syscall
951  *
952  * Hardware and (privileged) initialization code have arranged that by
953  * the time the sysenter instructions completes:
954  *
955  * - %rip is pointing to sys_sysenter (below).
956  * - %cs and %ss are set to kernel text and stack (data) selectors.
957  * - %rsp is pointing at the lwp's stack
958  * - interrupts have been disabled.
959  *
960  * Note that we are unable to return both "rvals" to userland with
```

```
 961  * this call, as %edx is used by the sysexit instruction.
 962  *
 963  * One final complication in this routine is its interaction with
 964  * single-stepping in a debugger.  For most of the system call mechanisms, the
 965  * CPU automatically clears the single-step flag before we enter the kernel.
 966  * The sysenter mechanism does not clear the flag, so a user single-stepping
 967  * through a libc routine may suddenly find themself single-stepping through the
 968  * kernel.  To detect this, kmdb and trap() both compare the trap %pc to the
 969  * [brand_]sys_enter addresses on each single-step trap.  If it finds that we
 970  * have single-stepped to a sysenter entry point, it explicitly clears the flag
 971  * and executes the sys_sysenter routine.
 938  * single-stepping in a debugger.  For most of the system call mechanisms,
 939  * the CPU automatically clears the single-step flag before we enter the
 940  * kernel.  The sysenter mechanism does not clear the flag, so a user
 941  * single-stepping through a libc routine may suddenly find themself
 942  * single-stepping through the kernel.  To detect this, kmdb compares the
 943  * trap %pc to the [brand_]sys_enter addresses on each single-step trap.
 944  * If it finds that we have single-stepped to a sysenter entry point, it
 945  * explicitly clears the flag and executes the sys_sysenter routine.
 972  *
 973  * One final complication in this final complication is the fact that we have
 974  * two different entry points for sysenter: brand_sys_sysenter and sys_sysenter.
 975  * If we enter at brand_sys_sysenter and start single-stepping through the
 976  * kernel with kmdb, we will eventually hit the instruction at sys_sysenter.
 977  * kmdb cannot distinguish between that valid single-step and the undesirable
 978  * one mentioned above.  To avoid this situation, we simply add a jump over the
 979  * instruction at sys_sysenter to make it impossible to single-step to it.
 947  * One final complication in this final complication is the fact that we
 948  * have two different entry points for sysenter: brand_sys_sysenter and
 949  * sys_sysenter.  If we enter at brand_sys_sysenter and start single-stepping
 950  * through the kernel with kmdb, we will eventually hit the instruction at
 951  * sys_sysenter.  kmdb cannot distinguish between that valid single-step
 952  * and the undesirable one mentioned above.  To avoid this situation, we
 953  * simply add a jump over the instruction at sys_sysenter to make it
 954  * impossible to single-step to it.
 980  */
 981 #if defined(__lint)

 983 void
 984 sys_sysenter()
 985 {}

 987 #else   /* __lint */

 989         ENTRY_NP(brand_sys_sysenter)
 990         SWAPGS                          /* kernel gsbase */
 991         ALTENTRY(_brand_sys_sysenter_post_swapgs)

 993         BRAND_CALLBACK(BRAND_CB_SYSENTER, BRAND_URET_FROM_REG(%rdx))
 994         /*
 995          * Jump over sys_sysenter to allow single-stepping as described
 996          * above.
 997          */
 998         jmp     _sys_sysenter_post_swapgs

1000         ALTENTRY(sys_sysenter)
1001         SWAPGS                          /* kernel gsbase */
1002         ALTENTRY(_sys_sysenter_post_swapgs)

 977         ALTENTRY(_sys_sysenter_post_swapgs)
1004         movq    %gs:CPU_THREAD, %r15

1006         movl    $U32CS_SEL, REGOFF_CS(%rsp)
1007         movl    %ecx, REGOFF_RSP(%rsp)          /* wrapper: %esp -> %ecx */
1008         movl    %edx, REGOFF_RIP(%rsp)          /* wrapper: %eip -> %edx */
1009         /*
```



```
1010          * NOTE: none of the instructions that run before we get here should
1011          * clobber bits in (R)FLAGS! This includes the kpti trampoline.
1012          */
1013         pushfq
1014         popq    %r10
1015         movl    $UDS_SEL, REGOFF_SS(%rsp)

1017         /*
1018          * Set the interrupt flag before storing the flags to the
1019          * flags image on the stack so we can return to user with
1020          * interrupts enabled if we return via sys_rtt_syscall32
1021          */
1022         orq     $PS_IE, %r10
1023         movq    %r10, REGOFF_RFL(%rsp)

1025         movl    %edi, REGOFF_RDI(%rsp)
1026         movl    %esi, REGOFF_RSI(%rsp)
1027         movl    %ebp, REGOFF_RBP(%rsp)
1028         movl    %ebx, REGOFF_RBX(%rsp)
1029         movl    %edx, REGOFF_RDX(%rsp)
1030         movl    %ecx, REGOFF_RCX(%rsp)
1031         movl    %eax, REGOFF_RAX(%rsp)          /* wrapper: sysc# -> %eax */
1032         movq    $0, REGOFF_SAVFP(%rsp)
1033         movq    $0, REGOFF_SAVPC(%rsp)

1035         /*
1036          * Copy these registers here in case we end up stopped with
1037          * someone (like, say, /proc) messing with our register state.
1038          * We don't -restore- them unless we have to in update_sregs.
1039          *
1040          * Since userland -can't- change fsbase or gsbase directly,
1041          * we don't bother to capture them here.
1042          */
1043         xorl    %ebx, %ebx
1044         movw    %ds, %bx
1045         movq    %rbx, REGOFF_DS(%rsp)
1046         movw    %es, %bx
1047         movq    %rbx, REGOFF_ES(%rsp)
1048         movw    %fs, %bx
1049         movq    %rbx, REGOFF_FS(%rsp)
1050         movw    %gs, %bx
1051         movq    %rbx, REGOFF_GS(%rsp)

1053         /*
1054          * If we're trying to use TRAPTRACE though, I take that back: we're
1055          * probably debugging some problem in the SWAPGS logic and want to know
1056          * what the incoming gsbase was.
1057          *
1058          * Since we already did SWAPGS, record the KGSBASE.
1059          */
1060 #if defined(DEBUG) && defined(TRAPTRACE) && !defined(__xpv)
1061         movl    $MSR_AMD_KGSBASE, %ecx
1062         rdmsr
1063         movl    %eax, REGOFF_GSBASE(%rsp)
1064         movl    %edx, REGOFF_GSBASE+4(%rsp)
1065 #endif

1067         /*
1068          * Application state saved in the regs structure on the stack
1069          * %eax is the syscall number
1070          * %rsp is the thread's stack, %r15 is curthread
1071          * REG_RSP(%rsp) is the user's stack
1072          */

1074         SYSCALL_TRAPTRACE($TT_SYSENTER)
```

```
1076         movq    %rsp, %rbp

1078         movq    T_LWP(%r15), %r14
1079         ASSERT_NO_RUPDATE_PENDING(%r14)

1081         ENABLE_INTR_FLAGS

1083         /*
1084          * Catch 64-bit process trying to issue sysenter instruction
1085          * on Nocona based systems.
1086          */
1087         movq    LWP_PROCP(%r14), %rax
1088         cmpq    $DATAMODEL_ILP32, P_MODEL(%rax)
1089         je      7f

1091         /*
1092          * For a non-32-bit process, simulate a #ud, since that's what
1093          * native hardware does.  The traptrace entry (above) will
1094          * let you know what really happened.
1095          */
1096         movq    $T_ILLINST, REGOFF_TRAPNO(%rsp)
1097         movq    REGOFF_CS(%rsp), %rdi
1098         movq    %rdi, REGOFF_ERR(%rsp)
1099         movq    %rsp, %rdi
1100         movq    REGOFF_RIP(%rsp), %rsi
1101         movl    %gs:CPU_ID, %edx
1102         call    trap
1103         jmp     _sys_rtt
1104 7:

1106         MSTATE_TRANSITION(LMS_USER, LMS_SYSTEM)
1107         movl    REGOFF_RAX(%rsp), %eax  /* (%rax damaged by mstate calls) */

1109         ASSERT_LWPTOREGS(%r14, %rsp)

1111         incq    %gs:CPU_STATS_SYS_SYSCALL

1113         /*
1114          * Make some space for MAXSYSARGS (currently 8) 32-bit args
1115          * placed into 64-bit (long) arg slots, plus one 64-bit
1116          * (long) arg count, maintaining 16 byte alignment.
1117          */
1118         subq    $SYS_DROP, %rsp
1119         movb    $LWP_SYS, LWP_STATE(%r14)
1120         movq    %r15, %rdi
1121         movq    %rsp, %rsi
1122         call    syscall_entry

1124         /*
1125          * Fetch the arguments copied onto the kernel stack and put
1126          * them in the right registers to invoke a C-style syscall handler.
1127          * %rax contains the handler address.
1128          */
1129         movq    %rax, %rbx
1130         movl    0(%rsp), %edi
1131         movl    8(%rsp), %esi
1132         movl    0x10(%rsp), %edx
1133         movl    0x18(%rsp), %ecx
1134         movl    0x20(%rsp), %r8d
1135         movl    0x28(%rsp), %r9d

1137         call    *SY_CALLC(%rbx)

1139         movq    %rbp, %rsp      /* pop the args */

1141         /*
```



```
1142          * amd64 syscall handlers -always- return a 64-bit value in %rax.
1143          * On the 32-bit kernel, the always return that value in %eax:%edx
1144          * as required by the 32-bit ABI.
1145          *
1146          * Simulate the same behaviour by unconditionally splitting the
1147          * return value in the same way.
1148          */
1149         movq    %rax, %r13
1150         shrq    $32, %r13       /* upper 32-bits into %edx */
1151         movl    %eax, %r12d     /* lower 32-bits into %eax */

1153         /*
1154          * Optimistically assume that there's no post-syscall
1155          * work to do.  (This is to avoid having to call syscall_mstate()
1156          * with interrupts disabled)
1157          */
1158         MSTATE_TRANSITION(LMS_SYSTEM, LMS_USER)

1160         /*
1161          * We must protect ourselves from being descheduled here;
1162          * If we were, and we ended up on another cpu, or another
1163          * lwp got int ahead of us, it could change the segment
1164          * registers without us noticing before we return to userland.
1165          *
1166          * This cli is undone in the tr_sysexit trampoline code.
1167          */
1168         cli
1169         CHECK_POSTSYS_NE(%r15, %r14, %ebx)
1170         jne     _full_syscall_postsys32
1171         SIMPLE_SYSCALL_POSTSYS(%r15, %r14, %bx)

1173         /*
1174          * To get back to userland, load up the 32-bit registers and
1175          * sysexit back where we came from.
1176          */

1178         /*
1179          * Interrupts will be turned on by the 'sti' executed just before
1180          * sysexit.  The following ensures that restoring the user's rflags
1181          * doesn't enable interrupts too soon.
1182          */
1183         andq    $_BITNOT(PS_IE), REGOFF_RFL(%rsp)

1185         /*
1186          * (There's no point in loading up %edx because the sysexit
1187          * mechanism smashes it.)
1188          */
1189         movl    %r12d, %eax
1190         movl    REGOFF_RBX(%rsp), %ebx
1191         movl    REGOFF_RBP(%rsp), %ebp
1192         movl    REGOFF_RSI(%rsp), %esi
1193         movl    REGOFF_RDI(%rsp), %edi

1195         movl    REGOFF_RIP(%rsp), %edx  /* sysexit: %edx -> %eip */
1196         pushq   REGOFF_RFL(%rsp)
1197         popfq
1198         movl    REGOFF_RSP(%rsp), %ecx  /* sysexit: %ecx -> %esp */
1199         ALTENTRY(sys_sysenter_swapgs_sysexit)
1200         jmp     tr_sysexit
1154         swapgs
1155         sti
1156         sysexit
1201         SET_SIZE(sys_sysenter_swapgs_sysexit)
______unchanged_portion_omitted_

1206 #endif  /* __lint */
```

```
1208 /*
1209  * This is the destination of the "int $T_SYSCALLINT" interrupt gate, used by
1210  * the generic i386 libc to do system calls. We do a small amount of setup
1211  * before jumping into the existing sys_syscall32 path.
1212  */
1213 #if defined(__lint)

1215 /*ARGSUSED*/
1216 void
1217 sys_syscall_int()
1218 {}

1220 #else   /* __lint */

1222         ENTRY_NP(brand_sys_syscall_int)
1223         SWAPGS                          /* kernel gsbase */
1224         XPV_TRAP_POP
1225         call    smap_enable
1226         BRAND_CALLBACK(BRAND_CB_INT91, BRAND_URET_FROM_INTR_STACK())
1227         jmp     nopop_syscall_int

1229         ALTENTRY(sys_syscall_int)
1230         SWAPGS                          /* kernel gsbase */
1231         XPV_TRAP_POP
1232         call    smap_enable

1234 nopop_syscall_int:
1235         movq    %gs:CPU_THREAD, %r15
1236         movq    T_STACK(%r15), %rsp
1237         movl    %eax, %eax
1238         /*
1239          * Set t_post_sys on this thread to force ourselves out via the slow
1240          * path. It might be possible at some later date to optimize this out
1241          * and use a faster return mechanism.
1242          */
1243         movb    $1, T_POST_SYS(%r15)
1244         CLEAN_CS
1245         jmp     _syscall32_save
1246         /*
1247          * There should be no instructions between this label and SWAPGS/IRET
1248          * or we could end up breaking branded zone support. See the usage of
1249          * this label in lx_brand_int80_callback and sn1_brand_int91_callback
1250          * for examples.
1251          *
1252          * We want to swapgs to maintain the invariant that all entries into
1253          * tr_iret_user are done on the user gsbase.
1254          */
1255         ALTENTRY(sys_sysint_swapgs_iret)
1256         SWAPGS
1257         jmp     tr_iret_user
1209         SWAPGS                          /* user gsbase */
1210         IRET
1258         /*NOTREACHED*/
1259         SET_SIZE(sys_sysint_swapgs_iret)
```

*unchanged_portion_omitted_*

```
**********************************************************
  137103 Fri Apr  6 17:25:01 2018
new/usr/src/uts/i86pc/os/cpuid.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2011, 2016 by Delphix. All rights reserved.
 * Copyright 2013 Nexenta Systems, Inc. All rights reserved.
 * Copyright 2014 Josef "Jeff" Sipek <jeffpc@josefsipek.net>
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */
/*
 * Portions Copyright 2009 Advanced Micro Devices, Inc.
 */
/*
 * Copyright 2018 Joyent, Inc.
 * Copyright 2017 Joyent, Inc.
 */
/*
 * Various routines to handle identification
 * and classification of x86 processors.
 */

#include <sys/types.h>
#include <sys/archsystm.h>
#include <sys/x86_archext.h>
#include <sys/kmem.h>
#include <sys/systm.h>
#include <sys/cmn_err.h>
#include <sys/sunddi.h>
#include <sys/sunndi.h>
#include <sys/cpuvar.h>
#include <sys/processor.h>
#include <sys/sysmacros.h>
#include <sys/pg.h>
#include <sys/fp.h>
#include <sys/controlregs.h>
#include <sys/bitmap.h>
#include <sys/auxv_386.h>
#include <sys/memnode.h>
```



```
#include <sys/pci_cfgspace.h>
#include <sys/comm_page.h>
#include <sys/mach_mmu.h>
#include <sys/tsc.h>

#ifdef __xpv
#include <sys/hypervisor.h>
#else
#include <sys/ontrap.h>
#endif

/*
 * Pass 0 of cpuid feature analysis happens in locore. It contains special code
 * to recognize Cyrix processors that are not cpuid-compliant, and to deal with
 * them accordingly. For most modern processors, feature detection occurs here
 * in pass 1.
 *
 * Pass 1 of cpuid feature analysis happens just at the beginning of mlsetup()
 * for the boot CPU and does the basic analysis that the early kernel needs.
 * x86_featureset is set based on the return value of cpuid_pass1() of the boot
 * CPU.
 *
 * Pass 1 includes:
 *
 *      o Determining vendor/model/family/stepping and setting x86_type and
 *        x86_vendor accordingly.
 *      o Processing the feature flags returned by the cpuid instruction while
 *        applying any workarounds or tricks for the specific processor.
 *      o Mapping the feature flags into illumos feature bits (X86_*).
 *      o Mapping the feature flags into Solaris feature bits (X86_*).
 *      o Processing extended feature flags if supported by the processor,
 *        again while applying specific processor knowledge.
 *      o Determining the CMT characteristics of the system.
 *
 * Pass 1 is done on non-boot CPUs during their initialization and the results
 * are used only as a meager attempt at ensuring that all processors within the
 * system support the same features.
 *
 * Pass 2 of cpuid feature analysis happens just at the beginning
 * of startup().  It just copies in and corrects the remainder
 * of the cpuid data we depend on: standard cpuid functions that we didn't
 * need for pass1 feature analysis, and extended cpuid functions beyond the
 * simple feature processing done in pass1.
 *
 * Pass 3 of cpuid analysis is invoked after basic kernel services; in
 * particular kernel memory allocation has been made available. It creates a
 * readable brand string based on the data collected in the first two passes.
 *
 * Pass 4 of cpuid analysis is invoked after post_startup() when all
 * the support infrastructure for various hardware features has been
 * initialized. It determines which processor features will be reported
 * to userland via the aux vector.
 *
 * All passes are executed on all CPUs, but only the boot CPU determines what
 * features the kernel will use.
 *
 * Much of the worst junk in this file is for the support of processors
 * that didn't really implement the cpuid instruction properly.
 *
 * NOTE: The accessor functions (cpuid_get*) are aware of, and ASSERT upon,
 * the pass numbers.  Accordingly, changes to the pass code may require changes
 * to the accessor code.
 */

uint_t x86_vendor = X86_VENDOR_IntelClone;
uint_t x86_type = X86_TYPE_OTHER;
```

```
124 uint_t x86_clflush_size = 0;

126 #if defined(__xpv)
127 int x86_use_pcid = 0;
128 int x86_use_invpcid = 0;
129 #else
130 int x86_use_pcid = -1;
131 int x86_use_invpcid = -1;
132 #endif

134 uint_t pentiumpro_bug4046376;

136 uchar_t x86_featureset[BT_SIZEOFMAP(NUM_X86_FEATURES)];

138 static char *x86_feature_names[NUM_X86_FEATURES] = {
139         "lgpg",
140         "tsc",
141         "msr",
142         "mtrr",
143         "pge",
144         "de",
145         "cmov",
146         "mmx",
147         "mca",
148         "pae",
149         "cv8",
150         "pat",
151         "sep",
152         "sse",
153         "sse2",
154         "htt",
155         "asysc",
156         "nx",
157         "sse3",
158         "cx16",
159         "cmp",
160         "tscp",
161         "mwait",
162         "sse4a",
163         "cpuid",
164         "ssse3",
165         "sse4_1",
166         "sse4_2",
167         "1gpg",
168         "clfsh",
169         "64",
170         "aes",
171         "pclmulqdq",
172         "xsave",
173         "avx",
174         "vmx",
175         "svm",
176         "topoext",
177         "f16c",
178         "rdrand",
179         "x2apic",
180         "avx2",
181         "bmi1",
182         "bmi2",
183         "fma",
184         "smep",
185         "smap",
186         "adx",
187         "rdseed",
188         "mpx",
189         "avx512f",
```



```
190         "avx512dq",
191         "avx512pf",
192         "avx512er",
193         "avx512cd",
194         "avx512bw",
195         "avx512vl",
196         "avx512fma",
197         "avx512vbmi",
198         "avx512_vpopcntdq",
199         "avx512_4vnniw",
200         "avx512_4fmaps",
201         "xsaveopt",
202         "xsavec",
203         "xsaves",
204         "sha",
205         "umip",
206         "pku",
207         "ospke",
208         "pcid",
209         "invpcid",
210 };
```

*unchanged_portion_omitted_*

```
 997 void
 998 cpuid_pass1(cpu_t *cpu, uchar_t *featureset)
 999 {
1000         uint32_t mask_ecx, mask_edx;
1001         struct cpuid_info *cpi;
1002         struct cpuid_regs *cp;
1003         int xcpuid;
1004 #if !defined(__xpv)
1005         extern int idle_cpu_prefer_mwait;
1006 #endif

1008         /*
1009          * Space statically allocated for BSP, ensure pointer is set
1010          */
1011         if (cpu->cpu_id == 0) {
1012                 if (cpu->cpu_m.mcpu_cpi == NULL)
1013                         cpu->cpu_m.mcpu_cpi = &cpuid_info0;
1014         }

1016         add_x86_feature(featureset, X86FSET_CPUID);

1018         cpi = cpu->cpu_m.mcpu_cpi;
1019         ASSERT(cpi != NULL);
1020         cp = &cpi->cpi_std[0];
1021         cp->cp_eax = 0;
1022         cpi->cpi_maxeax = __cpuid_insn(cp);
1023         {
1024                 uint32_t *iptr = (uint32_t *)cpi->cpi_vendorstr;
1025                 *iptr++ = cp->cp_ebx;
1026                 *iptr++ = cp->cp_edx;
1027                 *iptr++ = cp->cp_ecx;
1028                 *(char *)&cpi->cpi_vendorstr[12] = '\0';
1029         }

1031         cpi->cpi_vendor = _cpuid_vendorstr_to_vendorcode(cpi->cpi_vendorstr);
1032         x86_vendor = cpi->cpi_vendor; /* for compatibility */

1034         /*
1035          * Limit the range in case of weird hardware
1036          */
1037         if (cpi->cpi_maxeax > CPI_MAXEAX_MAX)
1038                 cpi->cpi_maxeax = CPI_MAXEAX_MAX;
1039         if (cpi->cpi_maxeax < 1)
```

```
1040                goto pass1_done;

1042        cp = &cpi->cpi_std[1];
1043        cp->cp_eax = 1;
1044        (void) __cpuid_insn(cp);

1046        /*
1047         * Extract identifying constants for easy access.
1048         */
1049        cpi->cpi_model = CPI_MODEL(cpi);
1050        cpi->cpi_family = CPI_FAMILY(cpi);

1052        if (cpi->cpi_family == 0xf)
1053                cpi->cpi_family += CPI_FAMILY_XTD(cpi);

1055        /*
1056         * Beware: AMD uses "extended model" iff base *FAMILY* == 0xf.
1057         * Intel, and presumably everyone else, uses model == 0xf, as
1058         * one would expect (max value means possible overflow).  Sigh.
1059         */

1061        switch (cpi->cpi_vendor) {
1062        case X86_VENDOR_Intel:
1063                if (IS_EXTENDED_MODEL_INTEL(cpi))
1064                        cpi->cpi_model += CPI_MODEL_XTD(cpi) << 4;
1065                break;
1066        case X86_VENDOR_AMD:
1067                if (CPI_FAMILY(cpi) == 0xf)
1068                        cpi->cpi_model += CPI_MODEL_XTD(cpi) << 4;
1069                break;
1070        default:
1071                if (cpi->cpi_model == 0xf)
1072                        cpi->cpi_model += CPI_MODEL_XTD(cpi) << 4;
1073                break;
1074        }

1076        cpi->cpi_step = CPI_STEP(cpi);
1077        cpi->cpi_brandid = CPI_BRANDID(cpi);

1079        /*
1080         * *default* assumptions:
1081         * - believe %edx feature word
1082         * - ignore %ecx feature word
1083         * - 32-bit virtual and physical addressing
1084         */
1085        mask_edx = 0xffffffff;
1086        mask_ecx = 0;

1088        cpi->cpi_pabits = cpi->cpi_vabits = 32;

1090        switch (cpi->cpi_vendor) {
1091        case X86_VENDOR_Intel:
1092                if (cpi->cpi_family == 5)
1093                        x86_type = X86_TYPE_P5;
1094                else if (IS_LEGACY_P6(cpi)) {
1095                        x86_type = X86_TYPE_P6;
1096                        pentiumpro_bug4046376 = 1;
1097                        /*
1098                         * Clear the SEP bit when it was set erroneously
1099                         */
1100                        if (cpi->cpi_model < 3 && cpi->cpi_step < 3)
1101                                cp->cp_edx &= ~CPUID_INTC_EDX_SEP;
1102                } else if (IS_NEW_F6(cpi) || cpi->cpi_family == 0xf) {
1103                        x86_type = X86_TYPE_P4;
1104                        /*
1105                         * We don't currently depend on any of the %ecx
```



```
1106                         * features until Prescott, so we'll only check
1107                         * this from P4 onwards.  We might want to revisit
1108                         * that idea later.
1109                         */
1110                        mask_ecx = 0xffffffff;
1111                } else if (cpi->cpi_family > 0xf)
1112                        mask_ecx = 0xffffffff;
1113                /*
1114                 * We don't support MONITOR/MWAIT if leaf 5 is not available
1115                 * to obtain the monitor linesize.
1116                 */
1117                if (cpi->cpi_maxeax < 5)
1118                        mask_ecx &= ~CPUID_INTC_ECX_MON;
1119                break;
1120        case X86_VENDOR_IntelClone:
1121        default:
1122                break;
1123        case X86_VENDOR_AMD:
1124 #if defined(OPTERON_ERRATUM_108)
1125                if (cpi->cpi_family == 0xf && cpi->cpi_model == 0xe) {
1126                        cp->cp_eax = (0xf0f & cp->cp_eax) | 0xc0;
1127                        cpi->cpi_model = 0xc;
1128                } else
1129 #endif
1130                if (cpi->cpi_family == 5) {
1131                        /*
1132                         * AMD K5 and K6
1133                         *
1134                         * These CPUs have an incomplete implementation
1135                         * of MCA/MCE which we mask away.
1136                         */
1137                        mask_edx &= ~(CPUID_INTC_EDX_MCE | CPUID_INTC_EDX_MCA);

1139                        /*
1140                         * Model 0 uses the wrong (APIC) bit
1141                         * to indicate PGE.  Fix it here.
1142                         */
1143                        if (cpi->cpi_model == 0) {
1144                                if (cp->cp_edx & 0x200) {
1145                                        cp->cp_edx &= ~0x200;
1146                                        cp->cp_edx |= CPUID_INTC_EDX_PGE;
1147                                }
1148                        }

1150                        /*
1151                         * Early models had problems w/ MMX; disable.
1152                         */
1153                        if (cpi->cpi_model < 6)
1154                                mask_edx &= ~CPUID_INTC_EDX_MMX;
1155                }

1157                /*
1158                 * For newer families, SSE3 and CX16, at least, are valid;
1159                 * enable all
1160                 */
1161                if (cpi->cpi_family >= 0xf)
1162                        mask_ecx = 0xffffffff;
1163                /*
1164                 * We don't support MONITOR/MWAIT if leaf 5 is not available
1165                 * to obtain the monitor linesize.
1166                 */
1167                if (cpi->cpi_maxeax < 5)
1168                        mask_ecx &= ~CPUID_INTC_ECX_MON;

1170 #if !defined(__xpv)
1171                /*
```

```
1172                 * Do not use MONITOR/MWAIT to halt in the idle loop on any AMD
1173                 * processors.  AMD does not intend MWAIT to be used in the cpu
1174                 * idle loop on current and future processors.  10h and future
1175                 * AMD processors use more power in MWAIT than HLT.
1176                 * Pre-family-10h Opterons do not have the MWAIT instruction.
1177                 */
1178                idle_cpu_prefer_mwait = 0;
1179 #endif

1181                break;
1182        case X86_VENDOR_TM:
1183                /*
1184                 * workaround the NT workaround in CMS 4.1
1185                 */
1186                if (cpi->cpi_family == 5 && cpi->cpi_model == 4 &&
1187                    (cpi->cpi_step == 2 || cpi->cpi_step == 3))
1188                        cp->cp_edx |= CPUID_INTC_EDX_CX8;
1189                break;
1190        case X86_VENDOR_Centaur:
1191                /*
1192                 * workaround the NT workarounds again
1193                 */
1194                if (cpi->cpi_family == 6)
1195                        cp->cp_edx |= CPUID_INTC_EDX_CX8;
1196                break;
1197        case X86_VENDOR_Cyrix:
1198                /*
1199                 * We rely heavily on the probing in locore
1200                 * to actually figure out what parts, if any,
1201                 * of the Cyrix cpuid instruction to believe.
1202                 */
1203                switch (x86_type) {
1204                case X86_TYPE_CYRIX_486:
1205                        mask_edx = 0;
1206                        break;
1207                case X86_TYPE_CYRIX_6x86:
1208                        mask_edx = 0;
1209                        break;
1210                case X86_TYPE_CYRIX_6x86L:
1211                        mask_edx =
1212                            CPUID_INTC_EDX_DE |
1213                            CPUID_INTC_EDX_CX8;
1214                        break;
1215                case X86_TYPE_CYRIX_6x86MX:
1216                        mask_edx =
1217                            CPUID_INTC_EDX_DE |
1218                            CPUID_INTC_EDX_MSR |
1219                            CPUID_INTC_EDX_CX8 |
1220                            CPUID_INTC_EDX_PGE |
1221                            CPUID_INTC_EDX_CMOV |
1222                            CPUID_INTC_EDX_MMX;
1223                        break;
1224                case X86_TYPE_CYRIX_GXm:
1225                        mask_edx =
1226                            CPUID_INTC_EDX_MSR |
1227                            CPUID_INTC_EDX_CX8 |
1228                            CPUID_INTC_EDX_CMOV |
1229                            CPUID_INTC_EDX_MMX;
1230                        break;
1231                case X86_TYPE_CYRIX_MediaGX:
1232                        break;
1233                case X86_TYPE_CYRIX_MII:
1234                case X86_TYPE_VIA_CYRIX_III:
1235                        mask_edx =
1236                            CPUID_INTC_EDX_DE |
1237                            CPUID_INTC_EDX_TSC |
```



```
1238                            CPUID_INTC_EDX_MSR |
1239                            CPUID_INTC_EDX_CX8 |
1240                            CPUID_INTC_EDX_PGE |
1241                            CPUID_INTC_EDX_CMOV |
1242                            CPUID_INTC_EDX_MMX;
1243                        break;
1244                default:
1245                        break;
1246                }
1247                break;
1248        }

1250 #if defined(__xpv)
1251        /*
1252         * Do not support MONITOR/MWAIT under a hypervisor
1253         */
1254        mask_ecx &= ~CPUID_INTC_ECX_MON;
1255        /*
1256         * Do not support XSAVE under a hypervisor for now
1257         */
1258        xsave_force_disable = B_TRUE;

1260 #endif  /* __xpv */

1262        if (xsave_force_disable) {
1263                mask_ecx &= ~CPUID_INTC_ECX_XSAVE;
1264                mask_ecx &= ~CPUID_INTC_ECX_AVX;
1265                mask_ecx &= ~CPUID_INTC_ECX_F16C;
1266                mask_ecx &= ~CPUID_INTC_ECX_FMA;
1267        }

1269        /*
1270         * Now we've figured out the masks that determine
1271         * which bits we choose to believe, apply the masks
1272         * to the feature words, then map the kernel's view
1273         * of these feature words into its feature word.
1274         */
1275        cp->cp_edx &= mask_edx;
1276        cp->cp_ecx &= mask_ecx;

1278        /*
1279         * apply any platform restrictions (we don't call this
1280         * immediately after __cpuid_insn here, because we need the
1281         * workarounds applied above first)
1282         */
1283        platform_cpuid_mangle(cpi->cpi_vendor, 1, cp);

1285        /*
1286         * In addition to ecx and edx, Intel is storing a bunch of instruction
1287         * set extensions in leaf 7's ebx, ecx, and edx.
1288         */
1289        if (cpi->cpi_vendor == X86_VENDOR_Intel && cpi->cpi_maxeax >= 7) {
1290                struct cpuid_regs *ecp;
1291                ecp = &cpi->cpi_std[7];
1292                ecp->cp_eax = 7;
1293                ecp->cp_ecx = 0;
1294                (void) __cpuid_insn(ecp);
1295                /*
1296                 * If XSAVE has been disabled, just ignore all of the
1297                 * extended-save-area dependent flags here.
1298                 */
1299                if (xsave_force_disable) {
1300                        ecp->cp_ebx &= ~CPUID_INTC_EBX_7_0_BMI1;
1301                        ecp->cp_ebx &= ~CPUID_INTC_EBX_7_0_BMI2;
1302                        ecp->cp_ebx &= ~CPUID_INTC_EBX_7_0_AVX2;
1303                        ecp->cp_ebx &= ~CPUID_INTC_EBX_7_0_MPX;
```

```
1304                        ecp->cp_ebx &= ~CPUID_INTC_EBX_7_0_ALL_AVX512;
1305                        ecp->cp_ecx &= ~CPUID_INTC_ECX_7_0_ALL_AVX512;
1306                        ecp->cp_edx &= ~CPUID_INTC_EDX_7_0_ALL_AVX512;
1307                }

1309                if (ecp->cp_ebx & CPUID_INTC_EBX_7_0_SMEP)
1310                        add_x86_feature(featureset, X86FSET_SMEP);

1312                if (ecp->cp_ebx & CPUID_INTC_EBX_7_0_INVPCID) {
1313                        add_x86_feature(featureset, X86FSET_INVPCID);
1314                }

1316                /*
1317                 * We check disable_smap here in addition to in startup_smap()
1318                 * to ensure CPUs that aren't the boot CPU don't accidentally
1319                 * include it in the feature set and thus generate a mismatched
1320                 * x86 feature set across CPUs. Note that at this time we only
1321                 * enable SMAP for the 64-bit kernel.
1322                 */
1323 #if defined(__amd64)
1324                if (ecp->cp_ebx & CPUID_INTC_EBX_7_0_SMAP &&
1325                    disable_smap == 0)
1326                        add_x86_feature(featureset, X86FSET_SMAP);
1327 #endif
1328                if (ecp->cp_ebx & CPUID_INTC_EBX_7_0_MPX)
1329                        add_x86_feature(featureset, X86FSET_MPX);

1331                if (ecp->cp_ebx & CPUID_INTC_EBX_7_0_RDSEED)
1332                        add_x86_feature(featureset, X86FSET_RDSEED);

1334                if (ecp->cp_ebx & CPUID_INTC_EBX_7_0_ADX)
1335                        add_x86_feature(featureset, X86FSET_ADX);
1336        }

1338        /*
1339         * fold in overrides from the "eeprom" mechanism
1340         */
1341        cp->cp_edx |= cpuid_feature_edx_include;
1342        cp->cp_edx &= ~cpuid_feature_edx_exclude;

1344        cp->cp_ecx |= cpuid_feature_ecx_include;
1345        cp->cp_ecx &= ~cpuid_feature_ecx_exclude;

1347        if (cp->cp_edx & CPUID_INTC_EDX_PSE) {
1348                add_x86_feature(featureset, X86FSET_LARGEPAGE);
1349        }
1350        if (cp->cp_edx & CPUID_INTC_EDX_TSC) {
1351                add_x86_feature(featureset, X86FSET_TSC);
1352        }
1353        if (cp->cp_edx & CPUID_INTC_EDX_MSR) {
1354                add_x86_feature(featureset, X86FSET_MSR);
1355        }
1356        if (cp->cp_edx & CPUID_INTC_EDX_MTRR) {
1357                add_x86_feature(featureset, X86FSET_MTRR);
1358        }
1359        if (cp->cp_edx & CPUID_INTC_EDX_PGE) {
1360                add_x86_feature(featureset, X86FSET_PGE);
1361        }
1362        if (cp->cp_edx & CPUID_INTC_EDX_CMOV) {
1363                add_x86_feature(featureset, X86FSET_CMOV);
1364        }
1365        if (cp->cp_edx & CPUID_INTC_EDX_MMX) {
1366                add_x86_feature(featureset, X86FSET_MMX);
1367        }
1368        if ((cp->cp_edx & CPUID_INTC_EDX_MCE) != 0 &&
1369            (cp->cp_edx & CPUID_INTC_EDX_MCA) != 0) {
```



```
1370                add_x86_feature(featureset, X86FSET_MCA);
1371        }
1372        if (cp->cp_edx & CPUID_INTC_EDX_PAE) {
1373                add_x86_feature(featureset, X86FSET_PAE);
1374        }
1375        if (cp->cp_edx & CPUID_INTC_EDX_CX8) {
1376                add_x86_feature(featureset, X86FSET_CX8);
1377        }
1378        if (cp->cp_ecx & CPUID_INTC_ECX_CX16) {
1379                add_x86_feature(featureset, X86FSET_CX16);
1380        }
1381        if (cp->cp_edx & CPUID_INTC_EDX_PAT) {
1382                add_x86_feature(featureset, X86FSET_PAT);
1383        }
1384        if (cp->cp_edx & CPUID_INTC_EDX_SEP) {
1385                add_x86_feature(featureset, X86FSET_SEP);
1386        }
1387        if (cp->cp_edx & CPUID_INTC_EDX_FXSR) {
1388                /*
1389                 * In our implementation, fxsave/fxrstor
1390                 * are prerequisites before we'll even
1391                 * try and do SSE things.
1392                 */
1393                if (cp->cp_edx & CPUID_INTC_EDX_SSE) {
1394                        add_x86_feature(featureset, X86FSET_SSE);
1395                }
1396                if (cp->cp_edx & CPUID_INTC_EDX_SSE2) {
1397                        add_x86_feature(featureset, X86FSET_SSE2);
1398                }
1399                if (cp->cp_ecx & CPUID_INTC_ECX_SSE3) {
1400                        add_x86_feature(featureset, X86FSET_SSE3);
1401                }
1402                if (cp->cp_ecx & CPUID_INTC_ECX_SSSE3) {
1403                        add_x86_feature(featureset, X86FSET_SSSE3);
1404                }
1405                if (cp->cp_ecx & CPUID_INTC_ECX_SSE4_1) {
1406                        add_x86_feature(featureset, X86FSET_SSE4_1);
1407                }
1408                if (cp->cp_ecx & CPUID_INTC_ECX_SSE4_2) {
1409                        add_x86_feature(featureset, X86FSET_SSE4_2);
1410                }
1411                if (cp->cp_ecx & CPUID_INTC_ECX_AES) {
1412                        add_x86_feature(featureset, X86FSET_AES);
1413                }
1414                if (cp->cp_ecx & CPUID_INTC_ECX_PCLMULQDQ) {
1415                        add_x86_feature(featureset, X86FSET_PCLMULQDQ);
1416                }

1418                if (cpi->cpi_std[7].cp_ebx & CPUID_INTC_EBX_7_0_SHA)
1419                        add_x86_feature(featureset, X86FSET_SHA);

1421                if (cpi->cpi_std[7].cp_ecx & CPUID_INTC_ECX_7_0_UMIP)
1422                        add_x86_feature(featureset, X86FSET_UMIP);
1423                if (cpi->cpi_std[7].cp_ecx & CPUID_INTC_ECX_7_0_PKU)
1424                        add_x86_feature(featureset, X86FSET_PKU);
1425                if (cpi->cpi_std[7].cp_ecx & CPUID_INTC_ECX_7_0_OSPKE)
1426                        add_x86_feature(featureset, X86FSET_OSPKE);

1428                if (cp->cp_ecx & CPUID_INTC_ECX_XSAVE) {
1429                        add_x86_feature(featureset, X86FSET_XSAVE);

1431                        /* We only test AVX & AVX512 when there is XSAVE */

1433                        if (cp->cp_ecx & CPUID_INTC_ECX_AVX) {
1434                                add_x86_feature(featureset,
1435                                    X86FSET_AVX);
```

```
1437                                    /*
1438                                     * Intel says we can't check these without also
1439                                     * checking AVX.
1440                                     */
1441                                    if (cp->cp_ecx & CPUID_INTC_ECX_F16C)
1442                                            add_x86_feature(featureset,
1443                                                X86FSET_F16C);
1444
1445                                    if (cp->cp_ecx & CPUID_INTC_ECX_FMA)
1446                                            add_x86_feature(featureset,
1447                                                X86FSET_FMA);
1448
1449                                    if (cpi->cpi_std[7].cp_ebx &
1450                                        CPUID_INTC_EBX_7_0_BMI1)
1451                                            add_x86_feature(featureset,
1452                                                X86FSET_BMI1);
1453
1454                                    if (cpi->cpi_std[7].cp_ebx &
1455                                        CPUID_INTC_EBX_7_0_BMI2)
1456                                            add_x86_feature(featureset,
1457                                                X86FSET_BMI2);
1458
1459                                    if (cpi->cpi_std[7].cp_ebx &
1460                                        CPUID_INTC_EBX_7_0_AVX2)
1461                                            add_x86_feature(featureset,
1462                                                X86FSET_AVX2);
1463                            }
1464
1465                            if (cpi->cpi_std[7].cp_ebx &
1466                                CPUID_INTC_EBX_7_0_AVX512F) {
1467                                    add_x86_feature(featureset, X86FSET_AVX512F);
1468
1469                                    if (cpi->cpi_std[7].cp_ebx &
1470                                        CPUID_INTC_EBX_7_0_AVX512DQ)
1471                                            add_x86_feature(featureset,
1472                                                X86FSET_AVX512DQ);
1473                                    if (cpi->cpi_std[7].cp_ebx &
1474                                        CPUID_INTC_EBX_7_0_AVX512IFMA)
1475                                            add_x86_feature(featureset,
1476                                                X86FSET_AVX512FMA);
1477                                    if (cpi->cpi_std[7].cp_ebx &
1478                                        CPUID_INTC_EBX_7_0_AVX512PF)
1479                                            add_x86_feature(featureset,
1480                                                X86FSET_AVX512PF);
1481                                    if (cpi->cpi_std[7].cp_ebx &
1482                                        CPUID_INTC_EBX_7_0_AVX512ER)
1483                                            add_x86_feature(featureset,
1484                                                X86FSET_AVX512ER);
1485                                    if (cpi->cpi_std[7].cp_ebx &
1486                                        CPUID_INTC_EBX_7_0_AVX512CD)
1487                                            add_x86_feature(featureset,
1488                                                X86FSET_AVX512CD);
1489                                    if (cpi->cpi_std[7].cp_ebx &
1490                                        CPUID_INTC_EBX_7_0_AVX512BW)
1491                                            add_x86_feature(featureset,
1492                                                X86FSET_AVX512BW);
1493                                    if (cpi->cpi_std[7].cp_ebx &
1494                                        CPUID_INTC_EBX_7_0_AVX512VL)
1495                                            add_x86_feature(featureset,
1496                                                X86FSET_AVX512VL);
1497
1498                                    if (cpi->cpi_std[7].cp_ecx &
1499                                        CPUID_INTC_ECX_7_0_AVX512VBMI)
1500                                            add_x86_feature(featureset,
1501                                                X86FSET_AVX512VBMI);
```



```
1502                                    if (cpi->cpi_std[7].cp_ecx &
1503                                        CPUID_INTC_ECX_7_0_AVX512VPOPCDQ)
1504                                            add_x86_feature(featureset,
1505                                                X86FSET_AVX512VPOPCDQ);
1506
1507                                    if (cpi->cpi_std[7].cp_edx &
1508                                        CPUID_INTC_EDX_7_0_AVX5124NNIW)
1509                                            add_x86_feature(featureset,
1510                                                X86FSET_AVX512NNIW);
1511                                    if (cpi->cpi_std[7].cp_edx &
1512                                        CPUID_INTC_EDX_7_0_AVX5124FMAPS)
1513                                            add_x86_feature(featureset,
1514                                                X86FSET_AVX512FMAPS);
1515                            }
1516                    }
1517            }
1518
1519            if (cpi->cpi_vendor == X86_VENDOR_Intel) {
1520                    if (cp->cp_ecx & CPUID_INTC_ECX_PCID) {
1521                            add_x86_feature(featureset, X86FSET_PCID);
1522                    }
1523            }
1524
1525            if (cp->cp_ecx & CPUID_INTC_ECX_X2APIC) {
1526                    add_x86_feature(featureset, X86FSET_X2APIC);
1527            }
1528            if (cp->cp_edx & CPUID_INTC_EDX_DE) {
1529                    add_x86_feature(featureset, X86FSET_DE);
1530            }
1531 #if !defined(__xpv)
1532            if (cp->cp_ecx & CPUID_INTC_ECX_MON) {
1533
1534                    /*
1535                     * We require the CLFLUSH instruction for erratum workaround
1536                     * to use MONITOR/MWAIT.
1537                     */
1538                    if (cp->cp_edx & CPUID_INTC_EDX_CLFSH) {
1539                            cpi->cpi_mwait.support |= MWAIT_SUPPORT;
1540                            add_x86_feature(featureset, X86FSET_MWAIT);
1541                    } else {
1542                            extern int idle_cpu_assert_cflush_monitor;
1543
1544                            /*
1545                             * All processors we are aware of which have
1546                             * MONITOR/MWAIT also have CLFLUSH.
1547                             */
1548                            if (idle_cpu_assert_cflush_monitor) {
1549                                    ASSERT((cp->cp_ecx & CPUID_INTC_ECX_MON) &&
1550                                        (cp->cp_edx & CPUID_INTC_EDX_CLFSH));
1551                            }
1552                    }
1553            }
1554 #endif  /* __xpv */
1555
1556            if (cp->cp_ecx & CPUID_INTC_ECX_VMX) {
1557                    add_x86_feature(featureset, X86FSET_VMX);
1558            }
1559
1560            if (cp->cp_ecx & CPUID_INTC_ECX_RDRAND)
1561                    add_x86_feature(featureset, X86FSET_RDRAND);
1562
1563            /*
1564             * Only need it first time, rest of the cpus would follow suit.
1565             * we only capture this for the bootcpu.
1566             */
1567            if (cp->cp_edx & CPUID_INTC_EDX_CLFSH) {
```

```
1568                    add_x86_feature(featureset, X86FSET_CLFSH);
1569                    x86_clflush_size = (BITX(cp->cp_ebx, 15, 8) * 8);
1570            }
1571            if (is_x86_feature(featureset, X86FSET_PAE))
1572                    cpi->cpi_pabits = 36;

1574            /*
1575             * Hyperthreading configuration is slightly tricky on Intel
1576             * and pure clones, and even trickier on AMD.
1577             *
1578             * (AMD chose to set the HTT bit on their CMP processors,
1579             * even though they're not actually hyperthreaded.  Thus it
1580             * takes a bit more work to figure out what's really going
1581             * on ... see the handling of the CMP_LGCY bit below)
1582             */
1583            if (cp->cp_edx & CPUID_INTC_EDX_HTT) {
1584                    cpi->cpi_ncpu_per_chip = CPI_CPU_COUNT(cpi);
1585                    if (cpi->cpi_ncpu_per_chip > 1)
1586                            add_x86_feature(featureset, X86FSET_HTT);
1587            } else {
1588                    cpi->cpi_ncpu_per_chip = 1;
1589            }

1591            if (cpi->cpi_vendor == X86_VENDOR_Intel && cpi->cpi_maxeax >= 0xD &&
1592                !xsave_force_disable) {
1593                    struct cpuid_regs r, *ecp;

1595                    ecp = &r;
1596                    ecp->cp_eax = 0xD;
1597                    ecp->cp_ecx = 1;
1598                    ecp->cp_edx = ecp->cp_ebx = 0;
1599                    (void) __cpuid_insn(ecp);

1601                    if (ecp->cp_eax & CPUID_INTC_EAX_D_1_XSAVEOPT)
1602                            add_x86_feature(featureset, X86FSET_XSAVEOPT);
1603                    if (ecp->cp_eax & CPUID_INTC_EAX_D_1_XSAVEC)
1604                            add_x86_feature(featureset, X86FSET_XSAVEC);
1605                    if (ecp->cp_eax & CPUID_INTC_EAX_D_1_XSAVES)
1606                            add_x86_feature(featureset, X86FSET_XSAVES);
1607            }

1609            /*
1610             * Work on the "extended" feature information, doing
1611             * some basic initialization for cpuid_pass2()
1612             */
1613            xcpuid = 0;
1614            switch (cpi->cpi_vendor) {
1615            case X86_VENDOR_Intel:
1616                    /*
1617                     * On KVM we know we will have proper support for extended
1618                     * cpuid.
1619                     */
1620                    if (IS_NEW_F6(cpi) || cpi->cpi_family >= 0xf ||
1621                        (get_hwenv() == HW_KVM && cpi->cpi_family == 6 &&
1622                        (cpi->cpi_model == 6 || cpi->cpi_model == 2)))
1623                            xcpuid++;
1624                    break;
1625            case X86_VENDOR_AMD:
1626                    if (cpi->cpi_family > 5 ||
1627                        (cpi->cpi_family == 5 && cpi->cpi_model >= 1))
1628                            xcpuid++;
1629                    break;
1630            case X86_VENDOR_Cyrix:
1631                    /*
1632                     * Only these Cyrix CPUs are -known- to support
1633                     * extended cpuid operations.
```



```
1634                     */
1635                    if (x86_type == X86_TYPE_VIA_CYRIX_III ||
1636                        x86_type == X86_TYPE_CYRIX_GXm)
1637                            xcpuid++;
1638                    break;
1639            case X86_VENDOR_Centaur:
1640            case X86_VENDOR_TM:
1641            default:
1642                    xcpuid++;
1643                    break;
1644            }

1646            if (xcpuid) {
1647                    cp = &cpi->cpi_extd[0];
1648                    cp->cp_eax = 0x80000000;
1649                    cpi->cpi_xmaxeax = __cpuid_insn(cp);
1650            }

1652            if (cpi->cpi_xmaxeax & 0x80000000) {

1654                    if (cpi->cpi_xmaxeax > CPI_XMAXEAX_MAX)
1655                            cpi->cpi_xmaxeax = CPI_XMAXEAX_MAX;

1657                    switch (cpi->cpi_vendor) {
1658                    case X86_VENDOR_Intel:
1659                    case X86_VENDOR_AMD:
1660                            if (cpi->cpi_xmaxeax < 0x80000001)
1661                                    break;
1662                            cp = &cpi->cpi_extd[1];
1663                            cp->cp_eax = 0x80000001;
1664                            (void) __cpuid_insn(cp);

1666                            if (cpi->cpi_vendor == X86_VENDOR_AMD &&
1667                                cpi->cpi_family == 5 &&
1668                                cpi->cpi_model == 6 &&
1669                                cpi->cpi_step == 6) {
1670                                    /*
1671                                     * K6 model 6 uses bit 10 to indicate SYSC
1672                                     * Later models use bit 11. Fix it here.
1673                                     */
1674                                    if (cp->cp_edx & 0x400) {
1675                                            cp->cp_edx &= ~0x400;
1676                                            cp->cp_edx |= CPUID_AMD_EDX_SYSC;
1677                                    }
1678                            }

1680                            platform_cpuid_mangle(cpi->cpi_vendor, 0x80000001, cp);

1682                            /*
1683                             * Compute the additions to the kernel's feature word.
1684                             */
1685                            if (cp->cp_edx & CPUID_AMD_EDX_NX) {
1686                                    add_x86_feature(featureset, X86FSET_NX);
1687                            }

1689                            /*
1690                             * Regardless whether or not we boot 64-bit,
1691                             * we should have a way to identify whether
1692                             * the CPU is capable of running 64-bit.
1693                             */
1694                            if (cp->cp_edx & CPUID_AMD_EDX_LM) {
1695                                    add_x86_feature(featureset, X86FSET_64);
1696                            }

1698 #if defined(__amd64)
1699                            /* 1 GB large page - enable only for 64 bit kernel */
```

```
1700                        if (cp->cp_edx & CPUID_AMD_EDX_1GPG) {
1701                                add_x86_feature(featureset, X86FSET_1GPG);
1702                        }
1703 #endif

1705                        if ((cpi->cpi_vendor == X86_VENDOR_AMD) &&
1706                            (cpi->cpi_std[1].cp_edx & CPUID_INTC_EDX_FXSR) &&
1707                            (cp->cp_ecx & CPUID_AMD_ECX_SSE4A)) {
1708                                add_x86_feature(featureset, X86FSET_SSE4A);
1709                        }

1711                        /*
1712                         * If both the HTT and CMP_LGCY bits are set,
1713                         * then we're not actually HyperThreaded.  Read
1714                         * "AMD CPUID Specification" for more details.
1715                         */
1716                        if (cpi->cpi_vendor == X86_VENDOR_AMD &&
1717                            is_x86_feature(featureset, X86FSET_HTT) &&
1718                            (cp->cp_ecx & CPUID_AMD_ECX_CMP_LGCY)) {
1719                                remove_x86_feature(featureset, X86FSET_HTT);
1720                                add_x86_feature(featureset, X86FSET_CMP);
1721                        }
1722 #if defined(__amd64)
1723                        /*
1724                         * It's really tricky to support syscall/sysret in
1725                         * the i386 kernel; we rely on sysenter/sysexit
1726                         * instead.  In the amd64 kernel, things are -way-
1727                         * better.
1728                         */
1729                        if (cp->cp_edx & CPUID_AMD_EDX_SYSC) {
1730                                add_x86_feature(featureset, X86FSET_ASYSC);
1731                        }

1733                        /*
1734                         * While we're thinking about system calls, note
1735                         * that AMD processors don't support sysenter
1736                         * in long mode at all, so don't try to program them.
1737                         */
1738                        if (x86_vendor == X86_VENDOR_AMD) {
1739                                remove_x86_feature(featureset, X86FSET_SEP);
1740                        }
1741 #endif
1742                        if (cp->cp_edx & CPUID_AMD_EDX_TSCP) {
1743                                add_x86_feature(featureset, X86FSET_TSCP);
1744                        }

1746                        if (cp->cp_ecx & CPUID_AMD_ECX_SVM) {
1747                                add_x86_feature(featureset, X86FSET_SVM);
1748                        }

1750                        if (cp->cp_ecx & CPUID_AMD_ECX_TOPOEXT) {
1751                                add_x86_feature(featureset, X86FSET_TOPOEXT);
1752                        }
1753                        break;
1754                default:
1755                        break;
1756                }

1758                /*
1759                 * Get CPUID data about processor cores and hyperthreads.
1760                 */
1761                switch (cpi->cpi_vendor) {
1762                case X86_VENDOR_Intel:
1763                        if (cpi->cpi_maxeax >= 4) {
1764                                cp = &cpi->cpi_std[4];
1765                                cp->cp_eax = 4;
```



```
1766                                cp->cp_ecx = 0;
1767                                (void) __cpuid_insn(cp);
1768                                platform_cpuid_mangle(cpi->cpi_vendor, 4, cp);
1769                        }
1770                        /*FALLTHROUGH*/
1771                case X86_VENDOR_AMD:
1772                        if (cpi->cpi_xmaxeax < 0x80000008)
1773                                break;
1774                        cp = &cpi->cpi_extd[8];
1775                        cp->cp_eax = 0x80000008;
1776                        (void) __cpuid_insn(cp);
1777                        platform_cpuid_mangle(cpi->cpi_vendor, 0x80000008, cp);

1779                        /*
1780                         * Virtual and physical address limits from
1781                         * cpuid override previously guessed values.
1782                         */
1783                        cpi->cpi_pabits = BITX(cp->cp_eax, 7, 0);
1784                        cpi->cpi_vabits = BITX(cp->cp_eax, 15, 8);
1785                        break;
1786                default:
1787                        break;
1788                }

1790                /*
1791                 * Derive the number of cores per chip
1792                 */
1793                switch (cpi->cpi_vendor) {
1794                case X86_VENDOR_Intel:
1795                        if (cpi->cpi_maxeax < 4) {
1796                                cpi->cpi_ncore_per_chip = 1;
1797                                break;
1798                        } else {
1799                                cpi->cpi_ncore_per_chip =
1800                                    BITX((cpi)->cpi_std[4].cp_eax, 31, 26) + 1;
1801                        }
1802                        break;
1803                case X86_VENDOR_AMD:
1804                        if (cpi->cpi_xmaxeax < 0x80000008) {
1805                                cpi->cpi_ncore_per_chip = 1;
1806                                break;
1807                        } else {
1808                                /*
1809                                 * On family 0xf cpuid fn 2 ECX[7:0] "NC" is
1810                                 * 1 less than the number of physical cores on
1811                                 * the chip.  In family 0x10 this value can
1812                                 * be affected by "downcoring" - it reflects
1813                                 * 1 less than the number of cores actually
1814                                 * enabled on this node.
1815                                 */
1816                                cpi->cpi_ncore_per_chip =
1817                                    BITX((cpi)->cpi_extd[8].cp_ecx, 7, 0) + 1;
1818                        }
1819                        break;
1820                default:
1821                        cpi->cpi_ncore_per_chip = 1;
1822                        break;
1823                }

1825                /*
1826                 * Get CPUID data about TSC Invariance in Deep C-State.
1827                 */
1828                switch (cpi->cpi_vendor) {
1829                case X86_VENDOR_Intel:
1830                        if (cpi->cpi_maxeax >= 7) {
1831                                cp = &cpi->cpi_extd[7];
```

```
1832                                cp->cp_eax = 0x80000007;
1833                                cp->cp_ecx = 0;
1834                                (void) __cpuid_insn(cp);
1835                        }
1836                        break;
1837                default:
1838                        break;
1839                }
1840        } else {
1841                cpi->cpi_ncore_per_chip = 1;
1842        }

1844        /*
1845         * If more than one core, then this processor is CMP.
1846         */
1847        if (cpi->cpi_ncore_per_chip > 1) {
1848                add_x86_feature(featureset, X86FSET_CMP);
1849        }

1851        /*
1852         * If the number of cores is the same as the number
1853         * of CPUs, then we cannot have HyperThreading.
1854         */
1855        if (cpi->cpi_ncpu_per_chip == cpi->cpi_ncore_per_chip) {
1856                remove_x86_feature(featureset, X86FSET_HTT);
1857        }

1859        cpi->cpi_apicid = CPI_APIC_ID(cpi);
1860        cpi->cpi_procnodes_per_pkg = 1;
1861        cpi->cpi_cores_per_compunit = 1;
1862        if (is_x86_feature(featureset, X86FSET_HTT) == B_FALSE &&
1863            is_x86_feature(featureset, X86FSET_CMP) == B_FALSE) {
1864                /*
1865                 * Single-core single-threaded processors.
1866                 */
1867                cpi->cpi_chipid = -1;
1868                cpi->cpi_clogid = 0;
1869                cpi->cpi_coreid = cpu->cpu_id;
1870                cpi->cpi_pkgcoreid = 0;
1871                if (cpi->cpi_vendor == X86_VENDOR_AMD)
1872                        cpi->cpi_procnodeid = BITX(cpi->cpi_apicid, 3, 0);
1873                else
1874                        cpi->cpi_procnodeid = cpi->cpi_chipid;
1875        } else if (cpi->cpi_ncpu_per_chip > 1) {
1876                if (cpi->cpi_vendor == X86_VENDOR_Intel)
1877                        cpuid_intel_getids(cpu, featureset);
1878                else if (cpi->cpi_vendor == X86_VENDOR_AMD)
1879                        cpuid_amd_getids(cpu);
1880                else {
1881                        /*
1882                         * All other processors are currently
1883                         * assumed to have single cores.
1884                         */
1885                        cpi->cpi_coreid = cpi->cpi_chipid;
1886                        cpi->cpi_pkgcoreid = 0;
1887                        cpi->cpi_procnodeid = cpi->cpi_chipid;
1888                        cpi->cpi_compunitid = cpi->cpi_chipid;
1889                }
1890        }

1892        /*
1893         * Synthesize chip "revision" and socket type
1894         */
1895        cpi->cpi_chiprev = _cpuid_chiprev(cpi->cpi_vendor, cpi->cpi_family,
1896            cpi->cpi_model, cpi->cpi_step);
1897        cpi->cpi_chiprevstr = _cpuid_chiprevstr(cpi->cpi_vendor,
```



```
1898            cpi->cpi_family, cpi->cpi_model, cpi->cpi_step);
1899        cpi->cpi_socket = _cpuid_skt(cpi->cpi_vendor, cpi->cpi_family,
1900            cpi->cpi_model, cpi->cpi_step);

1902        /*
1903         * While we're here, check for the AMD "Error Pointer Zero/Restore"
1904         * feature. This can be used to setup the FP save handlers
1905         * appropriately.
1906         */
1907        if (cpi->cpi_vendor == X86_VENDOR_AMD) {
1908                if (cpi->cpi_xmaxeax >= 0x80000008 &&
1909                    cpi->cpi_extd[8].cp_ebx & CPUID_AMD_EBX_ERR_PTR_ZERO) {
1910                        /* Special handling for AMD FP not necessary. */
1911                        cpi->cpi_fp_amd_save = 0;
1912                } else {
1913                        cpi->cpi_fp_amd_save = 1;
1914                }
1915        }

1917 pass1_done:
1918        cpi->cpi_pass = 1;
1919 }
```

*_____unchanged_portion_omitted_*

```
5017 void
5018 enable_pcid(void)
5019 {
5020        if (x86_use_pcid == -1)
5021                x86_use_pcid = is_x86_feature(x86_featureset, X86FSET_PCID);

5023        if (x86_use_invpcid == -1) {
5024                x86_use_invpcid = is_x86_feature(x86_featureset,
5025                    X86FSET_INVPCID);
5026        }

5028        if (!x86_use_pcid)
5029                return;

5031        /*
5032         * Intel say that on setting PCIDE, it immediately starts using the PCID
5033         * bits; better make sure there's nothing there.
5034         */
5035        ASSERT((getcr3() & MMU_PAGEOFFSET) == PCID_NONE);

5037        setcr4(getcr4() | CR4_PCIDE);
5038 }

5040 /*
5041  * Setup necessary registers to enable XSAVE feature on this processor.
5042  * This function needs to be called early enough, so that no xsave/xrstor
5043  * ops will execute on the processor before the MSRs are properly set up.
5044  *
5045  * Current implementation has the following assumption:
5046  * - cpuid_pass1() is done, so that X86 features are known.
5047  * - fpu_probe() is done, so that fp_save_mech is chosen.
5048  */
5049 void
5050 xsave_setup_msr(cpu_t *cpu)
5051 {
5052        ASSERT(fp_save_mech == FP_XSAVE);
5053        ASSERT(is_x86_feature(x86_featureset, X86FSET_XSAVE));

5055        /* Enable OSXSAVE in CR4. */
5056        setcr4(getcr4() | CR4_OSXSAVE);
5057        /*
5058         * Update SW copy of ECX, so that /dev/cpu/self/cpuid will report
```

```
5059          * correct value.
5060          */
5061         cpu->cpu_m.mcpu_cpi->cpi_std[1].cp_ecx |= CPUID_INTC_ECX_OSXSAVE;
5062         setup_xfem();
5063 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   66820 Fri Apr  6 17:25:01 2018
new/usr/src/uts/i86pc/os/fakebop.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 *
 * Copyright 2018 Joyent, Inc.  All rights reserved.
 * Copyright 2013 Joyent, Inc.  All rights reserved.
 */

/*
 * This file contains the functionality that mimics the boot operations
 * on SPARC systems or the old boot.bin/multiboot programs on x86 systems.
 * The x86 kernel now does everything on its own.
 */

#include <sys/types.h>
#include <sys/bootconf.h>
#include <sys/bootsvcs.h>
#include <sys/bootinfo.h>
#include <sys/multiboot.h>
#include <sys/multiboot2.h>
#include <sys/multiboot2_impl.h>
#include <sys/bootvfs.h>
#include <sys/bootprops.h>
#include <sys/varargs.h>
#include <sys/param.h>
#include <sys/machparam.h>
#include <sys/machsystm.h>
#include <sys/archsystm.h>
#include <sys/boot_console.h>
#include <sys/cmn_err.h>
#include <sys/systm.h>
#include <sys/promif.h>
#include <sys/archsystm.h>
#include <sys/x86_archext.h>
#include <sys/kobj.h>
```



```
#include <sys/privregs.h>
#include <sys/sysmacros.h>
#include <sys/ctype.h>
#include <sys/fastboot.h>
#ifdef __xpv
#include <sys/hypervisor.h>
#include <net/if.h>
#endif
#include <vm/kboot_mmu.h>
#include <vm/hat_pte.h>
#include <sys/kobj.h>
#include <sys/kobj_lex.h>
#include <sys/pci_cfgspace_impl.h>
#include <sys/fastboot_impl.h>
#include <sys/acpi/acconfig.h>
#include <sys/acpi/acpi.h>

static int have_console = 0;    /* set once primitive console is initialized */
static char *boot_args = "";

/*
 * Debugging macros
 */
static uint_t kbm_debug = 0;
#define DBG_MSG(s)      { if (kbm_debug) bop_printf(NULL, "%s", s); }
#define DBG(x)          { if (kbm_debug)                            \
        bop_printf(NULL, "%s is %" PRIx64 "\n", #x, (uint64_t)(x));     \
        }

#define PUT_STRING(s) {                         \
        char *cp;                               \
        for (cp = (s); *cp; ++cp)               \
                bcons_putchar(*cp);             \
        }

bootops_t bootop;       /* simple bootops we'll pass on to kernel */
struct bsys_mem bm;

/*
 * Boot info from "glue" code in low memory. xbootp is used by:
 *      do_bop_phys_alloc(), do_bsys_alloc() and boot_prop_finish().
 */
static struct xboot_info *xbootp;
static uintptr_t next_virt;     /* next available virtual address */
static paddr_t next_phys;       /* next available physical address from dboot */
static paddr_t high_phys = -(paddr_t)1; /* last used physical address */

/*
 * buffer for vsnprintf for console I/O
 */
#define BUFFERSIZE      512
static char buffer[BUFFERSIZE];

/*
 * stuff to store/report/manipulate boot property settings.
 */
typedef struct bootprop {
        struct bootprop *bp_next;
        char *bp_name;
        uint_t bp_vlen;
        char *bp_value;
} bootprop_t;
______unchanged_portion_omitted_
typedef int (*bios_func_t)(int, bios_regs_t *);

/*ARGSUSED*/
```

```
 839 static void
 840 do_bsys_doint(bootops_t *bop, int intnum, struct bop_regs *rp)
 841 {
 842 #if defined(__xpv)
 843         prom_panic("unsupported call to BOP_DOINT()\n");
 844 #else   /* __xpv */
 845         static int firsttime = 1;
 846         bios_func_t bios_func = (bios_func_t)(void *)(uintptr_t)0x5000;
 847         bios_regs_t br;

 849         /*
 850          * We're about to disable paging; we shouldn't be PCID enabled.
 851          */
 852         if (getcr4() & CR4_PCIDE)
 853                 prom_panic("do_bsys_doint() with PCID enabled\n");

 855         /*
 856          * The first time we do this, we have to copy the pre-packaged
 857          * low memory bios call code image into place.
 858          */
 859         if (firsttime) {
 860                 extern char bios_image[];
 861                 extern uint32_t bios_size;

 863                 bcopy(bios_image, (void *)bios_func, bios_size);
 864                 firsttime = 0;
 865         }

 867         br.ax = rp->eax.word.ax;
 868         br.bx = rp->ebx.word.bx;
 869         br.cx = rp->ecx.word.cx;
 870         br.dx = rp->edx.word.dx;
 871         br.bp = rp->ebp.word.bp;
 872         br.si = rp->esi.word.si;
 873         br.di = rp->edi.word.di;
 874         br.ds = rp->ds;
 875         br.es = rp->es;

 877         DBG_MSG("Doing BIOS call...");
 878         DBG(br.ax);
 879         DBG(br.bx);
 880         DBG(br.dx);
 881         rp->eflags = bios_func(intnum, &br);
 882         DBG_MSG("done\n");

 884         rp->eax.word.ax = br.ax;
 885         rp->ebx.word.bx = br.bx;
 886         rp->ecx.word.cx = br.cx;
 887         rp->edx.word.dx = br.dx;
 888         rp->ebp.word.bp = br.bp;
 889         rp->esi.word.si = br.si;
 890         rp->edi.word.di = br.di;
 891         rp->ds = br.ds;
 892         rp->es = br.es;
 893 #endif /* __xpv */
 894 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   57338 Fri Apr  6 17:25:02 2018
new/usr/src/uts/i86pc/os/intr.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2018 Joyent, Inc.  All rights reserverd.
 * Copyright (c) 2012, Joyent, Inc.  All rights reserverd.
 */

/*
 * To understand the present state of interrupt handling on i86pc, we must
 * first consider the history of interrupt controllers and our way of handling
 * interrupts.
 *
 * History of Interrupt Controllers on i86pc
 * -----------------------------------------
 *
 *    Intel 8259 and 8259A
 *
 * The first interrupt controller that attained widespread use on i86pc was
 * the Intel 8259(A) Programmable Interrupt Controller that first saw use with
 * the 8086. It took up to 8 interrupt sources and combined them into one
 * output wire. Up to 8 8259s could be slaved together providing up to 64 IRQs.
 * With the switch to the 8259A, level mode interrupts became possible. For a
 * long time on i86pc the 8259A was the only way to handle interrupts and it
 * had its own set of quirks. The 8259A and its corresponding interval timer
 * the 8254 are programmed using outb and inb instructions.
 *
 *    Intel Advanced Programmable Interrupt Controller (APIC)
 *
 * Starting around the time of the introduction of the P6 family
 * microarchitecture (i686) Intel introduced a new interrupt controller.
 * Instead of having the series of slaved 8259A devices, Intel opted to outfit
 * each processor with a Local APIC (lapic) and to outfit the system with at
 * least one, but potentially more, I/O APICs (ioapic). The lapics and ioapics
 * initially communicated over a dedicated bus, but this has since been
 * replaced. Each physical core and even hyperthread currently contains its
 * own local apic, which is not shared. There are a few exceptions for
 * hyperthreads, but that does not usually concern us.
 *
 * Instead of talking directly to 8259 for status, sending End Of Interrupt
```



```
 * (EOI), etc. a microprocessor now communicates directly to the lapic. This
 * also allows for each microprocessor to be able to have independent controls.
 * The programming method is different from the 8259. Consumers map the lapic
 * registers into uncacheable memory to read and manipulate the state.
 *
 * The number of addressable interrupt vectors was increased to 256. However
 * vectors 0-31 are reserved for the processor exception handling, leaving the
 * remaining vectors for general use. In addition to hardware generated
 * interrupts, the lapic provides a way for generating inter-processor
 * interrupts (IPI) which are the basis for CPU cross calls and CPU pokes.
 *
 * AMD ended up implementing the Intel APIC architecture in lieu of their work
 * with Cyrix.
 *
 *    Intel x2apic
 *
 * The x2apic is an extension to the lapic which started showing up around the
 * same time as the Sandy Bridge chipsets. It provides a new programming mode
 * as well as new features. The goal of the x2apic is to solve a few problems
 * with the previous generation of lapic and the x2apic is backwards compatible
 * with the previous programming and model. The only downsides to using the
 * backwards compatibility is that you are not able to take advantage of the new
 * x2apic features.
 *
 *    o The APIC ID is increased from an 8-bit value to a 32-bit value. This
 *    increases the maximum number of addressable physical processors beyond
 *    256. This new ID is assembled in a similar manner as the information that
 *    is obtainable by the extended cpuid topology leaves.
 *
 *    o A new means of generating IPIs was introduced.
 *
 *    o Instead of memory mapping the registers, the x2apic only allows for
 *    programming it through a series of wrmsrs. This has important semantic
 *    side effects. Recall that the registers were previously all mapped to
 *    uncachable memory which meant that all operations to the local apic were
 *    serializing instructions. With the switch to using wrmsrs this has been
 *    relaxed and these operations can no longer be assumed to be serializing
 *    instructions.
 *
 * Note for the rest of this we are only going to concern ourselves with the
 * apic and x2apic which practically all of i86pc has been using now for
 * quite some time.
 *
 * Interrupt Priority Levels
 * -------------------------
 *
 * On i86pc systems there are a total of fifteen interrupt priority levels
 * (ipls) which range from 1-15. Level 0 is for normal processing and
 * non-interrupt processing. To manipulate these values the family of spl
 * functions (which date back to UNIX on the PDP-11) are used. Specifically,
 * splr() to raise the priority level and splx() to lower it. One should not
 * generally call setspl() directly.
 *
 * Both i86pc and the supported SPARC platforms honor the same conventions for
 * the meaning behind these IPLs. The most important IPL is the platform's
 * LOCK_LEVEL (0xa on i86pc). If a thread is above LOCK_LEVEL it _must_ not
 * sleep on any synchronization object. The only allowed synchronization
 * primitive is a mutex that has been specifically initialized to be a spin
 * lock (see mutex_init(9F)). Another important level is DISP_LEVEL (0xb on
 * i86pc). You must be at DISP_LEVEL if you want to control the dispatcher.
 * The XC_HI_PIL is the highest level (0xf) and is used during cross-calls.
 *
 * Each interrupt that is registered in the system fires at a specific IPL.
 * Generally most interrupts fire below LOCK_LEVEL.
 *
 * PSM Drivers
```

```
 125  * -----------
 126  *
 127  * We currently have three sets of PSM (platform specific module) drivers
 128  * available. uppc, pcplusmp, and apix. uppc (uni-processor PC) is the original
 129  * driver that interacts with the 8259A and 8254. In general, it is not used
 130  * anymore given the prevalence of the apic.
 131  *
 132  * The system prefers to use the apix driver over the pcplusmp driver. The apix
 133  * driver requires HW support for an x2apic. If there is no x2apic HW, apix
 134  * will not be used. In general we prefer using the apix driver over the
 135  * pcplusmp driver because it gives us much more flexibility with respect to
 136  * interrupts. In the apix driver each local apic has its own independent set
 137  * of  interrupts, whereas the pcplusmp driver only has a single global set of
 138  * interrupts. This is why pcplusmp only supports a finite number of interrupts
 139  * per IPL -- generally 16, often less. The apix driver supports using either
 140  * the x2apic or the local apic programing modes. The programming mode does not
 141  * change the number of interrupts available, just the number of processors
 142  * that we can address. For the apix driver, the x2apic mode is enabled if the
 143  * system supports interrupt re-mapping, otherwise the module manages the
 144  * x2apic in local mode.
 145  *
 146  * When there is no x2apic present, we default back to the pcplusmp PSM driver.
 147  * In general, this is not problematic unless you have more than 256
 148  * processors in the machine or you do not have enough interrupts available.
 149  *
 150  * Controlling Interrupt Generation on i86pc
 151  * -----------------------------------------
 152  *
 153  * There are two different ways to manipulate which interrupts will be
 154  * generated on i86pc. Each offers different degrees of control.
 155  *
 156  * The first is through the flags register (eflags and rflags on i386 and amd64
 157  * respectively). The IF bit determines whether or not interrupts are enabled
 158  * or disabled. This is manipulated in one of several ways. The most common way
 159  * is through the cli and sti instructions. These clear the IF flag and set it,
 160  * respectively, for the current processor. The other common way is through the
 161  * use of the intr_clear and intr_restore functions.
 162  *
 163  * Assuming interrupts are not blocked by the IF flag, then the second form is
 164  * through the Processor-Priority Register (PPR). The PPR is used to determine
 165  * whether or not a pending interrupt should be delivered. If the ipl of the
 166  * new interrupt is higher than the current value in the PPR, then the lapic
 167  * will either deliver it immediately (if interrupts are not in progress) or it
 168  * will deliver it once the current interrupt processing has issued an EOI. The
 169  * highest unmasked interrupt will be the one delivered.
 170  *
 171  * The PPR register is based upon the max of the following two registers in the
 172  * lapic, the TPR register (also known as CR8 on amd64) that can be used to
 173  * mask interrupt levels, and the current vector. Because the pcplusmp module
 174  * always sets TPR appropriately early in the do_interrupt path, we can usually
 175  * just think that the PPR is the TPR. The pcplusmp module also issues an EOI
 176  * once it has set the TPR, so higher priority interrupts can come in while
 177  * we're servicing a lower priority interrupt.
 178  *
 179  * Handling Interrupts
 180  * -------------------
 181  *
 182  * Interrupts can be broken down into three categories based on priority and
 183  * source:
 184  *
 185  *   o High level interrupts
 186  *   o Low level hardware interrupts
 187  *   o Low level software interrupts
 188  *
 189  *   High Level Interrupts
 190  *
```



```
 191  * High level interrupts encompasses both hardware-sourced and software-sourced
 192  * interrupts. Examples of high level hardware interrupts include the serial
 193  * console. High level software-sourced interrupts are still delivered through
 194  * the local apic through IPIs. This is primarily cross calls.
 195  *
 196  * When a high level interrupt comes in, we will raise the SPL and then pin the
 197  * current lwp to the processor. We will use its lwp, but our own interrupt
 198  * stack and process the high level interrupt in-situ. These handlers are
 199  * designed to be very short in nature and cannot go to sleep, only block on a
 200  * spin lock. If the interrupt has a lot of work to do, it must generate a
 201  * low-priority software interrupt that will be processed later.
 202  *
 203  *   Low level hardware interrupts
 204  *
 205  * Low level hardware interrupts start off like their high-level cousins. The
 206  * current CPU contains a number of kernel threads (kthread_t) that can be used
 207  * to process low level interrupts. These are shared between both low level
 208  * hardware and software interrupts. Note that while we run with our
 209  * kthread_t, we borrow the pinned threads lwp_t until such a time as we hit a
 210  * synchronization object. If we hit one and need to sleep, then the scheduler
 211  * will instead create the rest of what we need.
 212  *
 213  *   Low level software interrupts
 214  *
 215  * Low level software interrupts are handled in a similar way as hardware
 216  * interrupts, but the notification vector is different. Each CPU has a bitmask
 217  * of pending software interrupts. We can notify a CPU to process software
 218  * interrupts through a specific trap vector as well as through several
 219  * checks that are performed throughout the code. These checks will look at
 220  * processing software interrupts as we lower our spl.
 221  *
 222  * We attempt to process the highest pending software interrupt that we can
 223  * which is greater than our current IPL. If none currently exist, then we move
 224  * on. We process a software interrupt in a similar fashion to a hardware
 225  * interrupt.
 226  *
 227  * Traditional Interrupt Flow
 228  * --------------------------
 229  *
 230  * The following diagram tracks the flow of the traditional uppc and pcplusmp
 231  * interrupt handlers. The apix driver has its own version of do_interrupt().
 232  * We come into the interrupt handler with all interrupts masked by the IF
 233  * flag. This is because we set up the handler using an interrupt-gate, which
 234  * is defined architecturally to have cleared the IF flag for us.
 235  *
 236  * +--------------+    +----------------+    +-----------+
 237  * | _interrupt() |--->| do_interrupt() |--->| *setlvl() |
 238  * +--------------+    +----------------+    +-----------+
 239  *                           |      |     |
 240  *                           |      |     |
 241  *              low-level|   |      |     | softint
 242  *                HW int |   |      |     +-----------------------------------------+
 243  * +--------------+      |   |      |                                               |
 244  * | intr_thread_ |<-----+   |      | hi-level int                                  |
 245  * | prolog()     |          |      +----------+                                    |
 246  * +--------------+          |     +--->| hilevel_ |      Not on intr stack        |
 247  *       |                   |           | intr_    |-----------------+             |
 248  *       |                   |           | prolog() |                 |             |
 249  * +------------+            |           +----------+                 |             |
 250  * | switch_sp_ |            |               | On intr                v             |
 251  * | and_call() |            |               | Stack          +------------+        |
 252  * +------------+            |               |                | switch_sp_ |        |
 253  *       |                   |               v                | and_call() |        |
 254  *       v                   |             +-----------+      +------------+        |
 255  * +-----------+             |             | dispatch_ |            |              |
 256  * | dispatch_ |   +---------------------| hilevel() |<-----------+              |
```

```
 257  * | hardint() |   |                  +-----------+                          |
 258  * +-----------+   |                                                         |
 259  *       |         v                                                         |
 260  *       |     +-----+  +----------------------+  +-----+  hi-level         |
 261  *       +---->| sti |->| av_dispatch_autovect |->| cli |---------+         |
 262  *             +-----+  +----------------------+  +-----+         |         |
 263  *                              |                 |               |         |
 264  *                              v                 |               |         |
 265  *                       +----------+             |               |         |
 266  *                       | for each |             |               |         |
 267  *                       | handler  |             |               |         |
 268  *                       |  *intr() |             |               v         |
 269  * +--------------+      +----------+             |      +----------------+ |
 270  * | intr_thread_ |                     low-level |      | hilevel_intr_  | |
 271  * | epilog()     |<-------------------------------+      | epilog()       | |
 272  * +--------------+                                       +----------------+ |
 273  *   |       |                                                  |           |
 274  *   |       +----------------------v     v---------------------+           |
 275  *   |                          +------------+                             |
 276  *   |   +--------------------->| *setlvlx() |                             |
 277  *   |   |                      +------------+                             |
 278  *   |   |                            |                                    |
 279  *   |   |                            v                                    |
 280  *   |   |      +--------+     +------------------+     +-------------+     |
 281  *   |   |      | return |<----| softint pending? |---->| dosoftint() |<----+
 282  *   |   |      +--------+  no +------------------+ yes +-------------+
 283  *   |   |           ^                                      |     |
 284  *   |   |           |  softint pil too low                 |     |
 285  *   |   |           +--------------------------------------+     |
 286  *   |   |                                                        v
 287  *   |   |    +-----------+     +------------+          +-----------+
 288  *   |   |    | dispatch_ |<----| switch_sp_ |<---------| *setspl() |
 289  *   |   |    | softint() |     | and_call() |          +-----------+
 290  *   |   |    +-----------+     +------------+
 291  *   |   |       |
 292  *   |   |       v
 293  *   |   |      +-----+  +----------------------+  +-----+  +------------+
 294  *   |   |      | sti |->| av_dispatch_autovect |->| cli |->| dosoftint_ |
 295  *   |   |      +-----+  +----------------------+  +-----+  | epilog()   |
 296  *   |   |                                                  +------------+
 297  *   |   |                                                     |     |
 298  *   |   +-----------------------------------------------------+     |
 299  *   v                                                               |
 300  * +-----------+                                                     |
 301  * | interrupt |                                                     |
 302  * | thread    |<----------------------------------------------------+
 303  * | blocked   |
 304  * +-----------+
 305  *      |
 306  *      v
 307  *  +----------------+  +------------+  +-----------+  +-------+  +---------+
 308  *  | set_base_spl() |->| *setlvlx() |->| splhigh() |->| sti() |->| swtch() |
 309  *  +----------------+  +------------+  +-----------+  +-------+  +---------+
 310  *
 311  *    Calls made on Interrupt Stacks and Epilogue routines
 312  *
 313  * We use the switch_sp_and_call() assembly routine to switch our sp to the
 314  * interrupt stacks and then call the appropriate dispatch function. In the
 315  * case of interrupts which may block, softints and hardints, we always ensure
 316  * that we are still on the interrupt thread when we call the epilog routine.
 317  * This is not just important, it's necessary. If the interrupt thread blocked,
 318  * we won't return from our switch_sp_and_call() function and instead we'll go
 319  * through and set ourselves up to swtch() directly.
 320  *
 321  * New Interrupt Flow
 322  * ------------------
```



```
 323  *
 324  * The apix module has its own interrupt path. This is done for various
 325  * reasons. The first is that rather than having global interrupt vectors, we
 326  * now have per-cpu vectors.
 327  *
 328  * The other substantial change is that the apix design does not use the TPR to
 329  * mask interrupts below the current level. In fact, except for one special
 330  * case, it does not use the TPR at all. Instead, it only uses the IF flag
 331  * (cli/sti) to either block all interrupts or allow any interrupts to come in.
 332  * The design is such that when interrupts are allowed to come in, if we are
 333  * currently servicing a higher priority interupt, the new interrupt is treated
 334  * as pending and serviced later. Specifically, in the pcplusmp module's
 335  * apic_intr_enter() the code masks interrupts at or below the current
 336  * IPL using the TPR before sending EOI, whereas the apix module's
 337  * apix_intr_enter() simply sends EOI.
 338  *
 339  * The one special case where the apix code uses the TPR is when it calls
 340  * through the apic_reg_ops function pointer apic_write_task_reg in
 341  * apix_init_intr() to initially mask all levels and then finally to enable all
 342  * levels.
 343  *
 344  * Recall that we come into the interrupt handler with all interrupts masked
 345  * by the IF flag. This is because we set up the handler using an
 346  * interrupt-gate which is defined architecturally to have cleared the IF flag
 347  * for us.
 348  *
 349  * +--------------+    +----------------------+
 350  * | _interrupt() |--->| apix_do_interrupt() |
 351  * +--------------+    +----------------------+
 352  *                              |
 353  *                hard int? +----+--------+ softint?
 354  *                          |             | (but no low-level looping)
 355  *                    +-----------+       |
 356  *                    | *setlvl() |       |
 357  * +---------+        +-----------+       +----------------------------------+
 358  * |apix_add_|    check IPL |                                                 |
 359  * |pending_ |<-------------+------+----------------------+                   |
 360  * |hardint()|        low-level int|          hi-level int|                   |
 361  * +---------+                    v                      v                   |
 362  *     | check IPL       +-----------------+     +---------------+           |
 363  *  +--+-----+           | apix_intr_      |     | apix_hilevel_ |           |
 364  *  |        |           | thread_prolog() |     | intr_prolog() |           |
 365  *  |      return        +-----------------+     +---------------+           |
 366  *  |                          |                     | On intr               |
 367  *  |                    +-------------+             | stack?  +-------------+  |
 368  *  |                    | switch_sp_ |              +---------| switch_sp_ |  |
 369  *  |                    | and_call() |                        | and_call() |  |
 370  *  |                    +-------------+                       +-------------+  |
 371  *  |                          |                               |              |
 372  *  |                    +-----------------+     +----------------+           |
 373  *  |                    | apix_dispatch_ |      | apix_dispatch_ |           |
 374  *  |                    | lowlevel()     |      | hilevel()      |           |
 375  *  |                    +-----------------+     +----------------+           |
 376  *  |                                |              |                        |
 377  *  |                                v              v                        |
 378  *  |                       +-------------------------+                       |
 379  *  |                       |apix_dispatch_by_vector()|----+                  |
 380  *  |                       +-------------------------+    |                  |
 381  *  |                !XC_HI_PIL|         |         |        |                  |
 382  *  |                       +---+   +-------+   +---+      |                  |
 383  *  |                       |sti|   |*intr()|   |cli|      |                  |
 384  *  |                       +---+   +-------+   +---+      |  hi-level?       |
 385  *  |                          +-----------------------------+----+            |
 386  *  |                          v                low-level?      v              |
 387  *  |                  +----------------+                +----------------+   |
 388  *  |                  | apix_intr_     |                | apix_hilevel_  |   |
```

```
 389  *  |                  | thread_epilog()|              | intr_epilog()  |   |
 390  *  |                  +----------------+              +----------------+   |
 391  *  |                           |                              |           |
 392  *  |        v------------------+------------------------------+           |
 393  *  |  +------------+                                                      |
 394  *  |  | *setlvlx() |   +--------------------------------------------------+
 395  *  |  +------------+   |
 396  *  |      |            |            +-------------------------------+ low
 397  *  v      v     v------+            v                               | level
 398  * +------------------+      +------------------+      +-----------+  | pending?
 399  * | apix_do_pending_ |----->| apix_do_pending_ |----->| apix_do_  |--+
 400  * | hilevel()        |      | hardint()        |      | softint() |  |
 401  * +------------------+      +------------------+      +-----------+    return
 402  *     |                       |                         |
 403  *     | while pending         | while pending           | while pending
 404  *     | hi-level              | low-level               | softint
 405  *     |                       |                         |
 406  *  +---------------+        +-----------------+       +-----------------+
 407  *  | apix_hilevel_ |        | apix_intr_      |       | apix_do_        |
 408  *  | intr_prolog() |        | thread_prolog() |       | softint_prolog()|
 409  *  +---------------+        +-----------------+       +-----------------+
 410  *     | On intr                     |                         |
 411  *     | stack? +------------+    +------------+        +------------+
 412  *     +--------| switch_sp_ |    | switch_sp_ |        | switch_sp_ |
 413  *     |        | and_call() |    | and_call() |        | and_call() |
 414  *     |        +------------+    +------------+        +------------+
 415  *     |           |                      |                      |
 416  *  +------------------+   +------------------+   +------------------------+
 417  *  | apix_dispatch_   |   | apix_dispatch_   |   | apix_dispatch_softint()|
 418  *  | pending_hilevel()|   | pending_hardint()|   +------------------------+
 419  *  +------------------+   +------------------+      |    |      |      |
 420  *    |          |           |          |            |    |      |      |
 421  *    | +----------------+  | +----------------+     |    |      |      |
 422  *    | | apix_hilevel_  |  | | apix_intr_     |     |    |      |      |
 423  *    | | intr_epilog()  |  | | thread_epilog()|     |    |      |      |
 424  *    | +----------------+  | +----------------+     |    |      |      |
 425  *    |          |          |       |                |    |      |      |
 426  *    |   +------------+    |  +----------+   +------+    |      |      |
 427  *    |   | *setlvlx() |    |  |*setlvlx()|   |           |      |      |
 428  *    |   +------------+    |  +----------+   |   +----------+   |   +---------+
 429  *    |                     |                 +---+ |av_       | +---+ |apix_do_ |
 430  * +--------------------------------+      |sti| |dispatch_ | |cli| |softint_ |
 431  * | apix_dispatch_pending_autovect()|      +---+ |softvect()| +---+ |epilog() |
 432  * +--------------------------------+            +----------+       +---------+
 433  *  |!XC_HI_PIL  |       |         |                 |
 434  * +---+  +-------+    +---+  +----------+         +-------+
 435  * |sti|  |*intr()|    |cli|  |apix_post_|         |*intr()|
 436  * +---+  +-------+    +---+  |hardint() |         +-------+
 437  *                            +----------+
 438  */

 440 #include <sys/cpuvar.h>
 441 #include <sys/cpu_event.h>
 442 #include <sys/regset.h>
 443 #include <sys/psw.h>
 444 #include <sys/types.h>
 445 #include <sys/thread.h>
 446 #include <sys/systm.h>
 447 #include <sys/segments.h>
 448 #include <sys/pcb.h>
 449 #include <sys/trap.h>
 450 #include <sys/ftrace.h>
 451 #include <sys/traptrace.h>
 452 #include <sys/clock.h>
 453 #include <sys/panic.h>
 454 #include <sys/disp.h>
```



```
 455 #include <vm/seg_kp.h>
 456 #include <sys/stack.h>
 457 #include <sys/sysmacros.h>
 458 #include <sys/cmn_err.h>
 459 #include <sys/kstat.h>
 460 #include <sys/smp_impldefs.h>
 461 #include <sys/pool_pset.h>
 462 #include <sys/zone.h>
 463 #include <sys/bitmap.h>
 464 #include <sys/archsystm.h>
 465 #include <sys/machsystm.h>
 466 #include <sys/ontrap.h>
 467 #include <sys/x86_archext.h>
 468 #include <sys/promif.h>
 469 #include <vm/hat_i86.h>
 470 #if defined(__xpv)
 471 #include <sys/hypervisor.h>
 472 #endif

 474 #if defined(__amd64) && !defined(__xpv)
 475 /* If this fails, then the padding numbers in machcpuvar.h are wrong. */
 476 CTASSERT((offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, mcpu_pad))
 477     < MMU_PAGESIZE);
 478 CTASSERT((offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, mcpu_kpti))
 479     >= MMU_PAGESIZE);
 480 CTASSERT((offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, mcpu_kpti_dbg))
 481     < 2 * MMU_PAGESIZE);
 482 CTASSERT((offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, mcpu_pad2))
 483     < 2 * MMU_PAGESIZE);
 484 CTASSERT(((sizeof (struct kpti_frame)) & 0xF) == 0);
 485 CTASSERT(((offsetof(cpu_t, cpu_m) + offsetof(struct machcpu, mcpu_kpti_dbg))
 486     & 0xF) == 0);
 487 CTASSERT((offsetof(struct kpti_frame, kf_tr_rsp) & 0xF) == 0);
 488 #endif

 490 #if defined(__xpv) && defined(DEBUG)

 492 /*
 493  * This panic message is intended as an aid to interrupt debugging.
 494  *
 495  * The associated assertion tests the condition of enabling
 496  * events when events are already enabled.  The implication
 497  * being that whatever code the programmer thought was
 498  * protected by having events disabled until the second
 499  * enable happened really wasn't protected at all ..
 500  */

 502 int stistipanic = 1;    /* controls the debug panic check */
 503 const char *stistimsg = "stisti";
 504 ulong_t laststi[NCPU];

 506 /*
 507  * This variable tracks the last place events were disabled on each cpu
 508  * it assists in debugging when asserts that interrupts are enabled trip.
 509  */
 510 ulong_t lastcli[NCPU];

 512 #endif

 514 void do_interrupt(struct regs *rp, trap_trace_rec_t *ttp);

 516 void (*do_interrupt_common)(struct regs *, trap_trace_rec_t *) = do_interrupt;
 517 uintptr_t (*get_intr_handler)(int, short) = NULL;

 519 /*
 520  * Set cpu's base SPL level to the highest active interrupt level
```

```
 521  */
 522 void
 523 set_base_spl(void)
 524 {
 525         struct cpu *cpu = CPU;
 526         uint16_t active = (uint16_t)cpu->cpu_intr_actv;

 528         cpu->cpu_base_spl = active == 0 ? 0 : bsrw_insn(active);
 529 }
______unchanged_portion_omitted_


1429 /*
1430  * Common tasks always done by _sys_rtt, called with interrupts disabled.
1431  * Returns 1 if returning to userland, 0 if returning to system mode.
1432  */
1433 int
1434 sys_rtt_common(struct regs *rp)
1435 {
1436         kthread_t *tp;
1437         extern void mutex_exit_critical_start();
1438         extern long mutex_exit_critical_size;
1439         extern void mutex_owner_running_critical_start();
1440         extern long mutex_owner_running_critical_size;

1442 loop:

1444         /*
1445          * Check if returning to user
1446          */
1447         tp = CPU->cpu_thread;
1448         if (USERMODE(rp->r_cs)) {
1449                 /*
1450                  * Check if AST pending.
1451                  */
1452                 if (tp->t_astflag) {
1453                         /*
1454                          * Let trap() handle the AST
1455                          */
1456                         sti();
1457                         rp->r_trapno = T_AST;
1458                         trap(rp, (caddr_t)0, CPU->cpu_id);
1459                         cli();
1460                         goto loop;
1461                 }

1463 #if defined(__amd64)
1464                 /*
1465                  * We are done if segment registers do not need updating.
1466                  */
1467                 if (tp->t_lwp->lwp_pcb.pcb_rupdate == 0)
1468                         return (1);

1470                 if (update_sregs(rp, tp->t_lwp)) {
1471                         /*
1472                          * 1 or more of the selectors is bad.
1473                          * Deliver a SIGSEGV.
1474                          */
1475                         proc_t *p = ttoproc(tp);

1477                         sti();
1478                         mutex_enter(&p->p_lock);
1479                         tp->t_lwp->lwp_cursig = SIGSEGV;
1480                         mutex_exit(&p->p_lock);
1481                         psig();
1482                         tp->t_sig_check = 1;
```



```
1483                         cli();
1484                 }
1485                 tp->t_lwp->lwp_pcb.pcb_rupdate = 0;

1487 #endif  /* __amd64 */
1488                 return (1);
1489         }

1491 #if !defined(__xpv)
1492         /*
1493          * Assert that we're not trying to return into the syscall return
1494          * trampolines. Things will go baaaaad if we try to do that.
1495          *
1496          * Note that none of these run with interrupts on, so this should
1497          * never happen (even in the sysexit case the STI doesn't take effect
1498          * until after sysexit finishes).
1499          */
1500         extern void tr_sysc_ret_start();
1501         extern void tr_sysc_ret_end();
1502         ASSERT(!(rp->r_pc >= (uintptr_t)tr_sysc_ret_start &&
1503             rp->r_pc <= (uintptr_t)tr_sysc_ret_end));
1504 #endif

1506         /*
1507          * Here if we are returning to supervisor mode.
1508          * Check for a kernel preemption request.
1509          */
1510         if (CPU->cpu_kprunrun && (rp->r_ps & PS_IE)) {

1512                 /*
1513                  * Do nothing if already in kpreempt
1514                  */
1515                 if (!tp->t_preempt_lk) {
1516                         tp->t_preempt_lk = 1;
1517                         sti();
1518                         kpreempt(1); /* asynchronous kpreempt call */
1519                         cli();
1520                         tp->t_preempt_lk = 0;
1521                 }
1522         }

1524         /*
1525          * If we interrupted the mutex_exit() critical region we must
1526          * reset the PC back to the beginning to prevent missed wakeups
1527          * See the comments in mutex_exit() for details.
1528          */
1529         if ((uintptr_t)rp->r_pc - (uintptr_t)mutex_exit_critical_start <
1530             mutex_exit_critical_size) {
1531                 rp->r_pc = (greg_t)mutex_exit_critical_start;
1532         }

1534         /*
1535          * If we interrupted the mutex_owner_running() critical region we
1536          * must reset the PC back to the beginning to prevent dereferencing
1537          * of a freed thread pointer. See the comments in mutex_owner_running
1538          * for details.
1539          */
1540         if ((uintptr_t)rp->r_pc -
1541             (uintptr_t)mutex_owner_running_critical_start <
1542             mutex_owner_running_critical_size) {
1543                 rp->r_pc = (greg_t)mutex_owner_running_critical_start;
1544         }

1546         return (0);
1547 }
______unchanged_portion_omitted_
```

```
**********************************************************
    4133 Fri Apr  6 17:25:02 2018
new/usr/src/uts/i86pc/os/mach_kdi.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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"

/*
 * Kernel/Debugger Interface (KDI) routines.  Called during debugger under
 * various system states (boot, while running, while the debugger has control).
 * Functions intended for use while the debugger has control may not grab any
 * locks or perform any functions that assume the availability of other system
 * services.
 */

#include <sys/systm.h>
#include <sys/x86_archext.h>
#include <sys/kdi_impl.h>
#include <sys/smp_impldefs.h>
#include <sys/psm_types.h>
#include <sys/segments.h>
#include <sys/archsystm.h>
#include <sys/controlregs.h>
#include <sys/trap.h>
#include <sys/kobj.h>
#include <sys/kobj_impl.h>
#include <sys/mach_mmu.h>

void
kdi_idt_write(gate_desc_t *gate, uint_t vec)
{
        gate_desc_t *idt = CPU->cpu_m.mcpu_idt;

        /*
         * See kdi_idtr_set().
         */
        if (idt == NULL) {
```



```
                desctbr_t idtr;
                rd_idtr(&idtr);
                idt = (gate_desc_t *)idtr.dtr_base;
        }

        idt[vec] = *gate;
}
_____unchanged_portion_omitted_

void
kdi_flush_caches(void)
{
        reload_cr3();
}

extern void kdi_slave_entry(void);

void
kdi_stop_slaves(int cpu, int doxc)
{
        if (doxc)
                kdi_xc_others(cpu, kdi_slave_entry);
}
_____unchanged_portion_omitted_
```

```
**********************************************************
   14906 Fri Apr  6 17:25:02 2018
new/usr/src/uts/i86pc/os/mlsetup.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2012 Gary Mills
 *
 * Copyright (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2011 by Delphix. All rights reserved.
 * Copyright 2018 Joyent, Inc.
 * Copyright 2016 Joyent, Inc.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/disp.h>
#include <sys/promif.h>
#include <sys/clock.h>
#include <sys/cpuvar.h>
#include <sys/stack.h>
#include <vm/as.h>
#include <vm/hat.h>
#include <sys/reboot.h>
#include <sys/avintr.h>
#include <sys/vtrace.h>
#include <sys/proc.h>
#include <sys/thread.h>
#include <sys/cpupart.h>
#include <sys/pset.h>
#include <sys/copyops.h>
#include <sys/pg.h>
#include <sys/disp.h>
#include <sys/debug.h>
#include <sys/sunddi.h>
#include <sys/x86_archext.h>
#include <sys/privregs.h>
#include <sys/machsystm.h>
#include <sys/ontrap.h>
#include <sys/bootconf.h>
```



```
#include <sys/boot_console.h>
#include <sys/kdi_machimpl.h>
#include <sys/archsystm.h>
#include <sys/promif.h>
#include <sys/pci_cfgspace.h>
#include <sys/bootvfs.h>
#include <sys/tsc.h>
#ifdef __xpv
#include <sys/hypervisor.h>
#else
#include <sys/xpv_support.h>
#endif

/*
 * some globals for patching the result of cpuid
 * to solve problems w/ creative cpu vendors
 */

extern uint32_t cpuid_feature_ecx_include;
extern uint32_t cpuid_feature_ecx_exclude;
extern uint32_t cpuid_feature_edx_include;
extern uint32_t cpuid_feature_edx_exclude;

/*
 * Set console mode
 */
static void
set_console_mode(uint8_t val)
{
        struct bop_regs rp = {0};

        rp.eax.byte.ah = 0x0;
        rp.eax.byte.al = val;
        rp.ebx.word.bx = 0x0;

        BOP_DOINT(bootops, 0x10, &rp);
}


/*
 * Setup routine called right before main(). Interposing this function
 * before main() allows us to call it in a machine-independent fashion.
 */
void
mlsetup(struct regs *rp)
{
        u_longlong_t prop_value;
        extern struct classfuncs sys_classfuncs;
        extern disp_t cpu0_disp;
        extern char t0stack[];
        extern int post_fastreboot;
        extern uint64_t plat_dr_options;

        ASSERT_STACK_ALIGNED();

        /*
         * initialize cpu_self
         */
        cpu[0]->cpu_self = cpu[0];

#if defined(__xpv)
        /*
         * Point at the hypervisor's virtual cpu structure
         */
        cpu[0]->cpu_m.mcpu_vcpu_info = &HYPERVISOR_shared_info->vcpu_info[0];
#endif
```

```
126         /*
127          * check if we've got special bits to clear or set
128          * when checking cpu features
129          */
131         if (bootprop_getval("cpuid_feature_ecx_include", &prop_value) != 0)
132                 cpuid_feature_ecx_include = 0;
133         else
134                 cpuid_feature_ecx_include = (uint32_t)prop_value;
136         if (bootprop_getval("cpuid_feature_ecx_exclude", &prop_value) != 0)
137                 cpuid_feature_ecx_exclude = 0;
138         else
139                 cpuid_feature_ecx_exclude = (uint32_t)prop_value;
141         if (bootprop_getval("cpuid_feature_edx_include", &prop_value) != 0)
142                 cpuid_feature_edx_include = 0;
143         else
144                 cpuid_feature_edx_include = (uint32_t)prop_value;
146         if (bootprop_getval("cpuid_feature_edx_exclude", &prop_value) != 0)
147                 cpuid_feature_edx_exclude = 0;
148         else
149                 cpuid_feature_edx_exclude = (uint32_t)prop_value;
151 #if !defined(__xpv)
152         /*
153          * Check to see if KPTI has been explicitly enabled or disabled.
154          * We have to check this before init_desctbls().
155          */
156         if (bootprop_getval("kpti", &prop_value) == 0) {
157                 kpti_enable = (uint64_t)(prop_value == 1);
158                 prom_printf("unix: forcing kpti to %s due to boot argument\n",
159                     (kpti_enable == 1) ? "ON" : "OFF");
160         } else {
161                 kpti_enable = 1;
162         }
164         if (bootprop_getval("pcid", &prop_value) == 0 && prop_value == 0) {
165                 prom_printf("unix: forcing pcid to OFF due to boot argument\n");
166                 x86_use_pcid = 0;
167         } else if (kpti_enable != 1) {
168                 x86_use_pcid = 0;
169         }
170 #endif
172         /*
173          * Initialize idt0, gdt0, ldt0_default, ktss0 and dftss.
174          */
175         init_desctbls();
177         /*
178          * lgrp_init() and possibly cpuid_pass1() need PCI config
179          * space access
180          */
181 #if defined(__xpv)
182         if (DOMAIN_IS_INITDOMAIN(xen_info))
183                 pci_cfgspace_init();
184 #else
185         pci_cfgspace_init();
186         /*
187          * Initialize the platform type from CPU 0 to ensure that
188          * determine_platform() is only ever called once.
189          */
190         determine_platform();
```



```
191 #endif
193         /*
194          * The first lightweight pass (pass0) through the cpuid data
195          * was done in locore before mlsetup was called.  Do the next
196          * pass in C code.
197          *
198          * The x86_featureset is initialized here based on the capabilities
199          * of the boot CPU.  Note that if we choose to support CPUs that have
200          * different feature sets (at which point we would almost certainly
201          * want to set the feature bits to correspond to the feature
202          * minimum) this value may be altered.
203          */
204         cpuid_pass1(cpu[0], x86_featureset);
206 #if !defined(__xpv)
207         if ((get_hwenv() & HW_XEN_HVM) != 0)
208                 xen_hvm_init();
210         /*
211          * Before we do anything with the TSCs, we need to work around
212          * Intel erratum BT81.  On some CPUs, warm reset does not
213          * clear the TSC.  If we are on such a CPU, we will clear TSC ourselves
214          * here.  Other CPUs will clear it when we boot them later, and the
215          * resulting skew will be handled by tsc_sync_master()/_slave();
216          * note that such skew already exists and has to be handled anyway.
217          *
218          * We do this only on metal.  This same problem can occur with a
219          * hypervisor that does not happen to virtualise a TSC that starts from
220          * zero, regardless of CPU type; however, we do not expect hypervisors
221          * that do not virtualise TSC that way to handle writes to TSC
222          * correctly, either.
223          */
224         if (get_hwenv() == HW_NATIVE &&
225             cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
226             cpuid_getfamily(CPU) == 6 &&
227             (cpuid_getmodel(CPU) == 0x2d || cpuid_getmodel(CPU) == 0x3e) &&
228             is_x86_feature(x86_featureset, X86FSET_TSC)) {
229                 (void) wrmsr(REG_TSC, 0UL);
230         }
232         /*
233          * Patch the tsc_read routine with appropriate set of instructions,
234          * depending on the processor family and architecure, to read the
235          * time-stamp counter while ensuring no out-of-order execution.
236          * Patch it while the kernel text is still writable.
237          *
238          * Note: tsc_read is not patched for intel processors whose family
239          * is >6 and for amd whose family >f (in case they don't support rdtscp
240          * instruction, unlikely). By default tsc_read will use cpuid for
241          * serialization in such cases. The following code needs to be
242          * revisited if intel processors of family >= f retains the
243          * instruction serialization nature of mfence instruction.
244          * Note: tsc_read is not patched for x86 processors which do
245          * not support "mfence". By default tsc_read will use cpuid for
246          * serialization in such cases.
247          *
248          * The Xen hypervisor does not correctly report whether rdtscp is
249          * supported or not, so we must assume that it is not.
250          */
251         if ((get_hwenv() & HW_XEN_HVM) == 0 &&
252             is_x86_feature(x86_featureset, X86FSET_TSCP))
253                 patch_tsc_read(TSC_TSCP);
254         else if (cpuid_getvendor(CPU) == X86_VENDOR_AMD &&
255             cpuid_getfamily(CPU) <= 0xf &&
256             is_x86_feature(x86_featureset, X86FSET_SSE2))
```

```
257                 patch_tsc_read(TSC_RDTSC_MFENCE);
258         else if (cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
259             cpuid_getfamily(CPU) <= 6 &&
260             is_x86_feature(x86_featureset, X86FSET_SSE2))
261                 patch_tsc_read(TSC_RDTSC_LFENCE);

263 #endif  /* !__xpv */

265 #if defined(__i386) && !defined(__xpv)
266         /*
267          * Some i386 processors do not implement the rdtsc instruction,
268          * or at least they do not implement it correctly. Patch them to
269          * return 0.
270          */
271         if (!is_x86_feature(x86_featureset, X86FSET_TSC))
272                 patch_tsc_read(TSC_NONE);
273 #endif  /* __i386 && !__xpv */

275 #if defined(__amd64) && !defined(__xpv)
276         patch_memops(cpuid_getvendor(CPU));
277 #endif  /* __amd64 && !__xpv */

279 #if !defined(__xpv)
280         /* XXPV what, if anything, should be dorked with here under xen? */

282         /*
283          * While we're thinking about the TSC, let's set up %cr4 so that
284          * userland can issue rdtsc, and initialize the TSC_AUX value
285          * (the cpuid) for the rdtscp instruction on appropriately
286          * capable hardware.
287          */
288         if (is_x86_feature(x86_featureset, X86FSET_TSC))
289                 setcr4(getcr4() & ~CR4_TSD);

291         if (is_x86_feature(x86_featureset, X86FSET_TSCP))
292                 (void) wrmsr(MSR_AMD_TSCAUX, 0);

294         /*
295          * Let's get the other %cr4 stuff while we're here. Note, we defer
296          * enabling CR4_SMAP until startup_end(); however, that's importantly
297          * before we start other CPUs. That ensures that it will be synced out
298          * to other CPUs.
299          */
300         if (is_x86_feature(x86_featureset, X86FSET_DE))
301                 setcr4(getcr4() | CR4_DE);

303         if (is_x86_feature(x86_featureset, X86FSET_SMEP))
304                 setcr4(getcr4() | CR4_SMEP);
305 #endif /* __xpv */

307         /*
308          * initialize t0
309          */
310         t0.t_stk = (caddr_t)rp - MINFRAME;
311         t0.t_stkbase = t0stack;
312         t0.t_pri = maxclsyspri - 3;
313         t0.t_schedflag = TS_LOAD | TS_DONT_SWAP;
314         t0.t_procp = &p0;
315         t0.t_plockp = &p0lock.pl_lock;
316         t0.t_lwp = &lwp0;
317         t0.t_forw = &t0;
318         t0.t_back = &t0;
319         t0.t_next = &t0;
320         t0.t_prev = &t0;
321         t0.t_cpu = cpu[0];
322         t0.t_disp_queue = &cpu0_disp;
```



```
323         t0.t_bind_cpu = PBIND_NONE;
324         t0.t_bind_pset = PS_NONE;
325         t0.t_bindflag = (uchar_t)default_binding_mode;
326         t0.t_cpupart = &cp_default;
327         t0.t_clfuncs = &sys_classfuncs.thread;
328         t0.t_copyops = NULL;
329         THREAD_ONPROC(&t0, CPU);

331         lwp0.lwp_thread = &t0;
332         lwp0.lwp_regs = (void *)rp;
333         lwp0.lwp_procp = &p0;
334         t0.t_tid = p0.p_lwpcnt = p0.p_lwprcnt = p0.p_lwpid = 1;

336         p0.p_exec = NULL;
337         p0.p_stat = SRUN;
338         p0.p_flag = SSYS;
339         p0.p_tlist = &t0;
340         p0.p_stksize = 2*PAGESIZE;
341         p0.p_stkpageszc = 0;
342         p0.p_as = &kas;
343         p0.p_lockp = &p0lock;
344         p0.p_brkpageszc = 0;
345         p0.p_t1_lgrpid = LGRP_NONE;
346         p0.p_tr_lgrpid = LGRP_NONE;
347         psecflags_default(&p0.p_secflags);

349         sigorset(&p0.p_ignore, &ignoredefault);

351         CPU->cpu_thread = &t0;
352         bzero(&cpu0_disp, sizeof (disp_t));
353         CPU->cpu_disp = &cpu0_disp;
354         CPU->cpu_disp->disp_cpu = CPU;
355         CPU->cpu_dispthread = &t0;
356         CPU->cpu_idle_thread = &t0;
357         CPU->cpu_flags = CPU_READY | CPU_RUNNING | CPU_EXISTS | CPU_ENABLE;
358         CPU->cpu_dispatch_pri = t0.t_pri;

360         CPU->cpu_id = 0;

362         CPU->cpu_pri = 12;              /* initial PIL for the boot CPU */

364         /*
365          * The kernel doesn't use LDTs unless a process explicitly requests one.
366          */
367         p0.p_ldt_desc = null_sdesc;

369         /*
370          * Initialize thread/cpu microstate accounting
371          */
372         init_mstate(&t0, LMS_SYSTEM);
373         init_cpu_mstate(CPU, CMS_SYSTEM);

375         /*
376          * Initialize lists of available and active CPUs.
377          */
378         cpu_list_init(CPU);

380         pg_cpu_bootstrap(CPU);

382         /*
383          * Now that we have taken over the GDT, IDT and have initialized
384          * active CPU list it's time to inform kmdb if present.
385          */
386         if (boothowto & RB_DEBUG)
387                 kdi_idt_sync();
```

```
389         if (BOP_GETPROPLEN(bootops, "efi-systab") < 0) {
390                 /*
391                  * In BIOS system, explicitly set console to text mode (0x3)
392                  * if this is a boot post Fast Reboot, and the console is set
393                  * to CONS_SCREEN_TEXT.
394                  */
395                 if (post_fastreboot &&
396                     boot_console_type(NULL) == CONS_SCREEN_TEXT) {
397                         set_console_mode(0x3);
398                 }
399         }

401         /*
402          * If requested (boot -d) drop into kmdb.
403          *
404          * This must be done after cpu_list_init() on the 64-bit kernel
405          * since taking a trap requires that we re-compute gsbase based
406          * on the cpu list.
407          */
408         if (boothowto & RB_DEBUGENTER)
409                 kmdb_enter();

411         cpu_vm_data_init(CPU);

413         rp->r_fp = 0;   /* terminate kernel stack traces! */

415         prom_init("kernel", (void *)NULL);

417         /* User-set option overrides firmware value. */
418         if (bootprop_getval(PLAT_DR_OPTIONS_NAME, &prop_value) == 0) {
419                 plat_dr_options = (uint64_t)prop_value;
420         }
421 #if defined(__xpv)
422         /* No support of DR operations on xpv */
423         plat_dr_options = 0;
424 #else   /* __xpv */
425         /* Flag PLAT_DR_FEATURE_ENABLED should only be set by DR driver. */
426         plat_dr_options &= ~PLAT_DR_FEATURE_ENABLED;
427 #ifndef __amd64
428         /* Only enable CPU/memory DR on 64 bits kernel. */
429         plat_dr_options &= ~PLAT_DR_FEATURE_MEMORY;
430         plat_dr_options &= ~PLAT_DR_FEATURE_CPU;
431 #endif  /* __amd64 */
432 #endif  /* __xpv */

434         /*
435          * Get value of "plat_dr_physmax" boot option.
436          * It overrides values calculated from MSCT or SRAT table.
437          */
438         if (bootprop_getval(PLAT_DR_PHYSMAX_NAME, &prop_value) == 0) {
439                 plat_dr_physmax = ((uint64_t)prop_value) >> PAGESHIFT;
440         }

442         /* Get value of boot_ncpus. */
443         if (bootprop_getval(BOOT_NCPUS_NAME, &prop_value) != 0) {
444                 boot_ncpus = NCPU;
445         } else {
446                 boot_ncpus = (int)prop_value;
447                 if (boot_ncpus <= 0 || boot_ncpus > NCPU)
448                         boot_ncpus = NCPU;
449         }

451         /*
452          * Set max_ncpus and boot_max_ncpus to boot_ncpus if platform doesn't
453          * support CPU DR operations.
454          */
```



```
455         if (plat_dr_support_cpu() == 0) {
456                 max_ncpus = boot_max_ncpus = boot_ncpus;
457         } else {
458                 if (bootprop_getval(PLAT_MAX_NCPUS_NAME, &prop_value) != 0) {
459                         max_ncpus = NCPU;
460                 } else {
461                         max_ncpus = (int)prop_value;
462                         if (max_ncpus <= 0 || max_ncpus > NCPU) {
463                                 max_ncpus = NCPU;
464                         }
465                         if (boot_ncpus > max_ncpus) {
466                                 boot_ncpus = max_ncpus;
467                         }
468                 }

470                 if (bootprop_getval(BOOT_MAX_NCPUS_NAME, &prop_value) != 0) {
471                         boot_max_ncpus = boot_ncpus;
472                 } else {
473                         boot_max_ncpus = (int)prop_value;
474                         if (boot_max_ncpus <= 0 || boot_max_ncpus > NCPU) {
475                                 boot_max_ncpus = boot_ncpus;
476                         } else if (boot_max_ncpus > max_ncpus) {
477                                 boot_max_ncpus = max_ncpus;
478                         }
479                 }
480         }

482         /*
483          * Initialize the lgrp framework
484          */
485         lgrp_init(LGRP_INIT_STAGE1);

487         if (boothowto & RB_HALT) {
488                 prom_printf("unix: kernel halted by -h flag\n");
489                 prom_enter_mon();
490         }

492         ASSERT_STACK_ALIGNED();

494         /*
495          * Fill out cpu_ucode_info.  Update microcode if necessary.
496          */
497         ucode_check(CPU);

499         if (workaround_errata(CPU) != 0)
500                 panic("critical workaround(s) missing for boot cpu");
501 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   18011 Fri Apr  6 17:25:02 2018
new/usr/src/uts/i86pc/os/mp_pc.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */
/*
 * Copyright 2018 Joyent, Inc
 * Copyright 2011 Joyent, Inc. All rights reserved.
 */

/*
 * Welcome to the world of the "real mode platter".
 * See also startup.c, mpcore.s and apic.c for related routines.
 */

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/cpuvar.h>
#include <sys/cpu_module.h>
#include <sys/kmem.h>
#include <sys/archsystm.h>
#include <sys/machsystm.h>
#include <sys/controlregs.h>
#include <sys/x86_archext.h>
#include <sys/smp_impldefs.h>
#include <sys/sysmacros.h>
#include <sys/mach_mmu.h>
#include <sys/promif.h>
#include <sys/cpu.h>
#include <sys/cpu_event.h>
#include <sys/sunndi.h>
#include <sys/fs/dv_node.h>
#include <vm/hat_i86.h>
#include <vm/as.h>

extern cpuset_t cpu_ready_set;
```



```
extern int  mp_start_cpu_common(cpu_t *cp, boolean_t boot);
extern void real_mode_start_cpu(void);
extern void real_mode_start_cpu_end(void);
extern void real_mode_stop_cpu_stage1(void);
extern void real_mode_stop_cpu_stage1_end(void);
extern void real_mode_stop_cpu_stage2(void);
extern void real_mode_stop_cpu_stage2_end(void);

void rmp_gdt_init(rm_platter_t *);

/*
 * Fill up the real mode platter to make it easy for real mode code to
 * kick it off. This area should really be one passed by boot to kernel
 * and guaranteed to be below 1MB and aligned to 16 bytes. Should also
 * have identical physical and virtual address in paged mode.
 */
static ushort_t *warm_reset_vector = NULL;

int
mach_cpucontext_init(void)
{
        ushort_t *vec;
        ulong_t addr;
        struct rm_platter *rm = (struct rm_platter *)rm_platter_va;

        if (!(vec = (ushort_t *)psm_map_phys(WARM_RESET_VECTOR,
            sizeof (vec), PROT_READ | PROT_WRITE)))
                return (-1);

        /*
         * setup secondary cpu bios boot up vector
         * Write page offset to 0x467 and page frame number to 0x469.
         */
        addr = (ulong_t)((caddr_t)rm->rm_code - (caddr_t)rm) + rm_platter_pa;
        vec[0] = (ushort_t)(addr & PAGEOFFSET);
        vec[1] = (ushort_t)((addr & (0xffffff & PAGEMASK)) >> 4);
        warm_reset_vector = vec;

        /* Map real mode platter into kas so kernel can access it. */
        hat_devload(kas.a_hat,
            (caddr_t)(uintptr_t)rm_platter_pa, MMU_PAGESIZE,
            btop(rm_platter_pa), PROT_READ | PROT_WRITE | PROT_EXEC,
            HAT_LOAD_NOCONSIST);

        /* Copy CPU startup code to rm_platter if it's still during boot. */
        if (!plat_dr_enabled()) {
                ASSERT((size_t)real_mode_start_cpu_end -
                    (size_t)real_mode_start_cpu <= RM_PLATTER_CODE_SIZE);
                bcopy((caddr_t)real_mode_start_cpu, (caddr_t)rm->rm_code,
                    (size_t)real_mode_start_cpu_end -
                    (size_t)real_mode_start_cpu);
        }

        return (0);
}
_____unchanged_portion_omitted_

#if defined(__amd64)
extern void *long_mode_64(void);
#endif  /* __amd64 */

/*ARGSUSED*/
void
rmp_gdt_init(rm_platter_t *rm)
{
```

```
#if defined(__amd64)
        /* Use the kas address space for the CPU startup thread. */
        if (mmu_ptob(kas.a_hat->hat_htable->ht_pfn) > 0xffffffffUL) {
        if (MAKECR3(kas.a_hat->hat_htable->ht_pfn) > 0xffffffffUL)
                panic("Cannot initialize CPUs; kernel's 64-bit page tables\n"
                    "located above 4G in physical memory (@ 0x%lx)",
                    mmu_ptob(kas.a_hat->hat_htable->ht_pfn));
        }
                    MAKECR3(kas.a_hat->hat_htable->ht_pfn));

        /*
         * Setup pseudo-descriptors for temporary GDT and IDT for use ONLY
         * by code in real_mode_start_cpu():
         *
         * GDT[0]:  NULL selector
         * GDT[1]:  64-bit CS: Long = 1, Present = 1, bits 12, 11 = 1
         *
         * Clear the IDT as interrupts will be off and a limit of 0 will cause
         * the CPU to triple fault and reset on an NMI, seemingly as reasonable
         * a course of action as any other, though it may cause the entire
         * platform to reset in some cases...
         */
        rm->rm_temp_gdt[0] = 0;
        rm->rm_temp_gdt[TEMPGDT_KCODE64] = 0x20980000000000ULL;

        rm->rm_temp_gdt_lim = (ushort_t)(sizeof (rm->rm_temp_gdt) - 1);
        rm->rm_temp_gdt_base = rm_platter_pa +
            (uint32_t)offsetof(rm_platter_t, rm_temp_gdt);
        rm->rm_temp_idt_lim = 0;
        rm->rm_temp_idt_base = 0;

        /*
         * Since the CPU needs to jump to protected mode using an identity
         * mapped address, we need to calculate it here.
         */
        rm->rm_longmode64_addr = rm_platter_pa +
            (uint32_t)((uintptr_t)long_mode_64 -
            (uintptr_t)real_mode_start_cpu);
#endif  /* __amd64 */
}

static void *
mach_cpucontext_alloc_tables(struct cpu *cp)
{
        tss_t *ntss;
        struct cpu_tables *ct;
        size_t ctsize;

        /*
         * Allocate space for stack, tss, gdt and idt. We round the size
         * allotted for cpu_tables up, so that the TSS is on a unique page.
         * This is more efficient when running in virtual machines.
         */
        ctsize = P2ROUNDUP(sizeof (*ct), PAGESIZE);
        ct = kmem_zalloc(ctsize, KM_SLEEP);
        ct = kmem_zalloc(P2ROUNDUP(sizeof (*ct), PAGESIZE), KM_SLEEP);
        if ((uintptr_t)ct & PAGEOFFSET)
                panic("mach_cpucontext_alloc_tables: cpu%d misaligned tables",
                    cp->cpu_id);

        ntss = cp->cpu_tss = &ct->ct_tss;

#if defined(__amd64)
        uintptr_t va;
        size_t len;
```



```
        /*
         * #DF (double fault).
         */
        ntss->tss_ist1 = (uintptr_t)&ct->ct_stack1[sizeof (ct->ct_stack1)];
        ntss->tss_ist1 = (uint64_t)&ct->ct_stack[sizeof (ct->ct_stack)];

        /*
         * #NM (non-maskable interrupt)
         */
        ntss->tss_ist2 = (uintptr_t)&ct->ct_stack2[sizeof (ct->ct_stack2)];

        /*
         * #MC (machine check exception / hardware error)
         */
        ntss->tss_ist3 = (uintptr_t)&ct->ct_stack3[sizeof (ct->ct_stack3)];

        /*
         * #DB, #BP debug interrupts and KDI/kmdb
         */
        ntss->tss_ist4 = (uintptr_t)&cp->cpu_m.mcpu_kpti_dbg.kf_tr_rsp;

        if (kpti_enable == 1) {
                /*
                 * #GP, #PF, #SS fault interrupts
                 */
                ntss->tss_ist5 = (uintptr_t)&cp->cpu_m.mcpu_kpti_flt.kf_tr_rsp;

                /*
                 * Used by all other interrupts
                 */
                ntss->tss_ist6 = (uint64_t)&cp->cpu_m.mcpu_kpti.kf_tr_rsp;

                /*
                 * On AMD64 we need to make sure that all of the pages of the
                 * struct cpu_tables are punched through onto the user CPU for
                 * kpti.
                 *
                 * The final page will always be the TSS, so treat that
                 * separately.
                 */
                for (va = (uintptr_t)ct, len = ctsize - MMU_PAGESIZE;
                    len >= MMU_PAGESIZE;
                    len -= MMU_PAGESIZE, va += MMU_PAGESIZE) {
                        /* The doublefault stack must be RW */
                        hati_cpu_punchin(cp, va, PROT_READ | PROT_WRITE);
                }
                ASSERT3U((uintptr_t)ntss, ==, va);
                hati_cpu_punchin(cp, (uintptr_t)ntss, PROT_READ);
        }

#elif defined(__i386)

        ntss->tss_esp0 = ntss->tss_esp1 = ntss->tss_esp2 = ntss->tss_esp =
            (uint32_t)&ct->ct_stack1[sizeof (ct->ct_stack1)];
            (uint32_t)&ct->ct_stack[sizeof (ct->ct_stack)];

        ntss->tss_ss0 = ntss->tss_ss1 = ntss->tss_ss2 = ntss->tss_ss = KDS_SEL;

        ntss->tss_eip = (uint32_t)cp->cpu_thread->t_pc;

        ntss->tss_cs = KCS_SEL;
        ntss->tss_ds = ntss->tss_es = KDS_SEL;
        ntss->tss_fs = KFS_SEL;
        ntss->tss_gs = KGS_SEL;

#endif  /* __i386 */
```

```
262         /*
263          * Set I/O bit map offset equal to size of TSS segment limit
264          * for no I/O permission map. This will cause all user I/O
265          * instructions to generate #gp fault.
266          */
267         ntss->tss_bitmapbase = sizeof (*ntss);

269         /*
270          * Setup kernel tss.
271          */
272         set_syssegd((system_desc_t *)&cp->cpu_gdt[GDT_KTSS], cp->cpu_tss,
273             sizeof (*cp->cpu_tss) - 1, SDT_SYSTSS, SEL_KPL);

275         return (ct);
276 }

278 void *
279 mach_cpucontext_xalloc(struct cpu *cp, int optype)
280 {
281         size_t len;
282         struct cpu_tables *ct;
283         rm_platter_t *rm = (rm_platter_t *)rm_platter_va;
284         static int cpu_halt_code_ready;

286         if (optype == MACH_CPUCONTEXT_OP_STOP) {
287                 ASSERT(plat_dr_enabled());

289                 /*
290                  * The WARM_RESET_VECTOR has a limitation that the physical
291                  * address written to it must be page-aligned. To work around
292                  * this limitation, the CPU stop code has been splitted into
293                  * two stages.
294                  * The stage 2 code, which implements the real logic to halt
295                  * CPUs, is copied to the rm_cpu_halt_code field in the real
296                  * mode platter. The stage 1 code, which simply jumps to the
297                  * stage 2 code in the rm_cpu_halt_code field, is copied to
298                  * rm_code field in the real mode platter and it may be
299                  * overwritten after the CPU has been stopped.
300                  */
301                 if (!cpu_halt_code_ready) {
302                         /*
303                          * The rm_cpu_halt_code field in the real mode platter
304                          * is used by the CPU stop code only. So only copy the
305                          * CPU stop stage 2 code into the rm_cpu_halt_code
306                          * field on the first call.
307                          */
308                         len = (size_t)real_mode_stop_cpu_stage2_end -
309                             (size_t)real_mode_stop_cpu_stage2;
310                         ASSERT(len <= RM_PLATTER_CPU_HALT_CODE_SIZE);
311                         bcopy((caddr_t)real_mode_stop_cpu_stage2,
312                             (caddr_t)rm->rm_cpu_halt_code, len);
313                         cpu_halt_code_ready = 1;
314                 }

316                 /*
317                  * The rm_code field in the real mode platter is shared by
318                  * the CPU start, CPU stop, CPR and fast reboot code. So copy
319                  * the CPU stop stage 1 code into the rm_code field every time.
320                  */
321                 len = (size_t)real_mode_stop_cpu_stage1_end -
322                     (size_t)real_mode_stop_cpu_stage1;
323                 ASSERT(len <= RM_PLATTER_CODE_SIZE);
324                 bcopy((caddr_t)real_mode_stop_cpu_stage1,
325                     (caddr_t)rm->rm_code, len);
326                 rm->rm_cpu_halted = 0;
```



```
328                 return (cp->cpu_m.mcpu_mach_ctx_ptr);
329         } else if (optype != MACH_CPUCONTEXT_OP_START) {
330                 return (NULL);
331         }

333         /*
334          * Only need to allocate tables when starting CPU.
335          * Tables allocated when starting CPU will be reused when stopping CPU.
336          */
337         ct = mach_cpucontext_alloc_tables(cp);
338         if (ct == NULL) {
339                 return (NULL);
340         }

342         /* Copy CPU startup code to rm_platter for CPU hot-add operations. */
343         if (plat_dr_enabled()) {
344                 bcopy((caddr_t)real_mode_start_cpu, (caddr_t)rm->rm_code,
345                     (size_t)real_mode_start_cpu_end -
346                     (size_t)real_mode_start_cpu);
347         }

349         /*
350          * Now copy all that we've set up onto the real mode platter
351          * for the real mode code to digest as part of starting the cpu.
352          */
353         rm->rm_idt_base = cp->cpu_idt;
354         rm->rm_idt_lim = sizeof (*cp->cpu_idt) * NIDT - 1;
355         rm->rm_gdt_base = cp->cpu_gdt;
356         rm->rm_gdt_lim = sizeof (*cp->cpu_gdt) * NGDT - 1;

358         /*
359          * CPU needs to access kernel address space after powering on.
311          * When hot-adding CPU at runtime, directly use top level page table
312          * of kas other than the return value of getcr3(). getcr3() returns
313          * current process's top level page table, which may be different from
314          * the one of kas.
360          */
361         rm->rm_pdbr = MAKECR3(kas.a_hat->hat_htable->ht_pfn, PCID_NONE);
316         rm->rm_pdbr = MAKECR3(kas.a_hat->hat_htable->ht_pfn);
362         rm->rm_cpu = cp->cpu_id;

364         /*
365          * We need to mask off any bits set on our boot CPU that can't apply
366          * while the subject CPU is initializing.  If appropriate, they are
367          * enabled later on.
320          * For hot-adding CPU at runtime, Machine Check and Performance Counter
321          * should be disabled. They will be enabled on demand after CPU powers
322          * on successfully
368          */
369         rm->rm_cr4 = getcr4();
370         rm->rm_cr4 &= ~(CR4_MCE | CR4_PCE | CR4_PCIDE);
325         rm->rm_cr4 &= ~(CR4_MCE | CR4_PCE);

372         rmp_gdt_init(rm);

374         return (ct);
375 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
   51721 Fri Apr  6 17:25:02 2018
new/usr/src/uts/i86pc/os/mp_startup.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */
/*
 * Copyright 2018 Joyent, Inc.
 * Copyright 2016 Joyent, Inc.
 * Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
 */

#include <sys/types.h>
#include <sys/thread.h>
#include <sys/cpuvar.h>
#include <sys/cpu.h>
#include <sys/t_lock.h>
#include <sys/param.h>
#include <sys/proc.h>
#include <sys/disp.h>
#include <sys/class.h>
#include <sys/cmn_err.h>
#include <sys/debug.h>
#include <sys/note.h>
#include <sys/asm_linkage.h>
#include <sys/x_call.h>
#include <sys/systm.h>
#include <sys/var.h>
#include <sys/vtrace.h>
#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg_kmem.h>
#include <vm/seg_kp.h>
#include <sys/segments.h>
#include <sys/kmem.h>
#include <sys/stack.h>
#include <sys/smp_impldefs.h>
```



```
#include <sys/x86_archext.h>
#include <sys/machsystm.h>
#include <sys/traptrace.h>
#include <sys/clock.h>
#include <sys/cpc_impl.h>
#include <sys/pg.h>
#include <sys/cmt.h>
#include <sys/dtrace.h>
#include <sys/archsystm.h>
#include <sys/fp.h>
#include <sys/reboot.h>
#include <sys/kdi_machimpl.h>
#include <vm/hat_i86.h>
#include <vm/vm_dep.h>
#include <sys/memnode.h>
#include <sys/pci_cfgspace.h>
#include <sys/mach_mmu.h>
#include <sys/sysmacros.h>
#if defined(__xpv)
#include <sys/hypervisor.h>
#endif
#include <sys/cpu_module.h>
#include <sys/ontrap.h>

struct cpu      cpus[1] __aligned(MMU_PAGESIZE);
struct cpu      *cpu[NCPU] = {&cpus[0]};
struct cpu      *cpu_free_list;
cpu_core_t      cpu_core[NCPU];
struct cpu      cpus[1];                        /* CPU data */
struct cpu      *cpu[NCPU] = {&cpus[0]};        /* pointers to all CPUs */
struct cpu      *cpu_free_list;                 /* list for released CPUs */
cpu_core_t      cpu_core[NCPU];                 /* cpu_core structures */

#define cpu_next_free   cpu_prev

/*
 * Useful for disabling MP bring-up on a MP capable system.
 */
int use_mp = 1;

/*
 * to be set by a PSM to indicate what cpus
 * are sitting around on the system.
 */
cpuset_t mp_cpus;

/*
 * This variable is used by the hat layer to decide whether or not
 * critical sections are needed to prevent race conditions.  For sun4m,
 * this variable is set once enough MP initialization has been done in
 * order to allow cross calls.
 */
int flushes_require_xcalls;

cpuset_t cpu_ready_set;         /* initialized in startup() */

static void mp_startup_boot(void);
static void mp_startup_hotplug(void);

static void cpu_sep_enable(void);
static void cpu_sep_disable(void);
static void cpu_asysc_enable(void);
static void cpu_asysc_disable(void);

/*
 * Init CPU info - get CPU type info for processor_info system call.
```

```
 */
void
init_cpu_info(struct cpu *cp)
{
        processor_info_t *pi = &cp->cpu_type_info;

        /*
         * Get clock-frequency property for the CPU.
         */
        pi->pi_clock = cpu_freq;

        /*
         * Current frequency in Hz.
         */
        cp->cpu_curr_clock = cpu_freq_hz;

        /*
         * Supported frequencies.
         */
        if (cp->cpu_supp_freqs == NULL) {
                cpu_set_supp_freqs(cp, NULL);
        }

        (void) strcpy(pi->pi_processor_type, "i386");
        if (fpu_exists)
                (void) strcpy(pi->pi_fputypes, "i387 compatible");

        cp->cpu_idstr = kmem_zalloc(CPU_IDSTRLEN, KM_SLEEP);
        cp->cpu_brandstr = kmem_zalloc(CPU_IDSTRLEN, KM_SLEEP);

        /*
         * If called for the BSP, cp is equal to current CPU.
         * For non-BSPs, cpuid info of cp is not ready yet, so use cpuid info
         * of current CPU as default values for cpu_idstr and cpu_brandstr.
         * They will be corrected in mp_startup_common() after cpuid_pass1()
         * has been invoked on target CPU.
         */
        (void) cpuid_getidstr(CPU, cp->cpu_idstr, CPU_IDSTRLEN);
        (void) cpuid_getbrandstr(CPU, cp->cpu_brandstr, CPU_IDSTRLEN);
}

/*
 * Configure syscall support on this CPU.
 */
/*ARGSUSED*/
void
init_cpu_syscall(struct cpu *cp)
{
        kpreempt_disable();

#if defined(__amd64)
        if (is_x86_feature(x86_featureset, X86FSET_MSR) &&
            is_x86_feature(x86_featureset, X86FSET_ASYSC)) {
                uint64_t flags;

#if !defined(__xpv)
#if !defined(__lint)
                /*
                 * The syscall instruction imposes a certain ordering on
                 * segment selectors, so we double-check that ordering
                 * here.
                 */
                CTASSERT(KDS_SEL == KCS_SEL + 8);
                CTASSERT(UDS_SEL == U32CS_SEL + 8);
                CTASSERT(UCS_SEL == U32CS_SEL + 16);
                ASSERT(KDS_SEL == KCS_SEL + 8);
```



```
                ASSERT(UDS_SEL == U32CS_SEL + 8);
                ASSERT(UCS_SEL == U32CS_SEL + 16);
#endif

                /*
                 * Turn syscall/sysret extensions on.
                 */
                cpu_asysc_enable();

                /*
                 * Program the magic registers ..
                 */
                wrmsr(MSR_AMD_STAR,
                    ((uint64_t)(U32CS_SEL << 16 | KCS_SEL)) << 32);
                if (kpti_enable == 1) {
                        wrmsr(MSR_AMD_LSTAR,
                            (uint64_t)(uintptr_t)tr_sys_syscall);
                        wrmsr(MSR_AMD_CSTAR,
                            (uint64_t)(uintptr_t)tr_sys_syscall32);
                } else {
                        wrmsr(MSR_AMD_LSTAR,
                            (uint64_t)(uintptr_t)sys_syscall);
                        wrmsr(MSR_AMD_CSTAR,
                            (uint64_t)(uintptr_t)sys_syscall32);
                }
                wrmsr(MSR_AMD_LSTAR, (uint64_t)(uintptr_t)sys_syscall);
                wrmsr(MSR_AMD_CSTAR, (uint64_t)(uintptr_t)sys_syscall32);

                /*
                 * This list of flags is masked off the incoming
                 * %rfl when we enter the kernel.
                 */
                flags = PS_IE | PS_T;
                if (is_x86_feature(x86_featureset, X86FSET_SMAP) == B_TRUE)
                        flags |= PS_ACHK;
                wrmsr(MSR_AMD_SFMASK, flags);
        }
#endif

        /*
         * On 32-bit kernels, we use sysenter/sysexit because it's too
         * hard to use syscall/sysret, and it is more portable anyway.
         *
         * On 64-bit kernels on Nocona machines, the 32-bit syscall
         * variant isn't available to 32-bit applications, but sysenter is.
         */
        if (is_x86_feature(x86_featureset, X86FSET_MSR) &&
            is_x86_feature(x86_featureset, X86FSET_SEP)) {

#if !defined(__xpv)
#if !defined(__lint)
                /*
                 * The sysenter instruction imposes a certain ordering on
                 * segment selectors, so we double-check that ordering
                 * here. See "sysenter" in Intel document 245471-012, "IA-32
                 * Intel Architecture Software Developer's Manual Volume 2:
                 * Instruction Set Reference"
                 */
                CTASSERT(KDS_SEL == KCS_SEL + 8);
                ASSERT(KDS_SEL == KCS_SEL + 8);

                CTASSERT(U32CS_SEL == ((KCS_SEL + 16) | 3));
                CTASSERT(UDS_SEL == U32CS_SEL + 8);
                ASSERT32(UCS_SEL == ((KCS_SEL + 16) | 3));
                ASSERT32(UDS_SEL == UCS_SEL + 8);
```

```
233                 ASSERT64(U32CS_SEL == ((KCS_SEL + 16) | 3));
234                 ASSERT64(UDS_SEL == U32CS_SEL + 8);
237 #endif

239                 cpu_sep_enable();

241                 /*
242                  * resume() sets this value to the base of the threads stack
243                  * via a context handler.
244                  */
245                 wrmsr(MSR_INTC_SEP_ESP, 0);

247                 if (kpti_enable == 1) {
248                         wrmsr(MSR_INTC_SEP_EIP,
249                             (uint64_t)(uintptr_t)tr_sys_sysenter);
250                 } else {
251                         wrmsr(MSR_INTC_SEP_EIP,
252                             (uint64_t)(uintptr_t)sys_sysenter);
244                 wrmsr(MSR_INTC_SEP_EIP, (uint64_t)(uintptr_t)sys_sysenter);
253                 }
254         }

256         kpreempt_enable();
257 }
_____unchanged_portion_omitted_
275 #endif /* !defined(__xpv) */

277 /*
278  * Multiprocessor initialization.
279  *
280  * Allocate and initialize the cpu structure, TRAPTRACE buffer, and the
281  * startup and idle threads for the specified CPU.
282  * Parameter boot is true for boot time operations and is false for CPU
283  * DR operations.
284  */
285 static struct cpu *
286 mp_cpu_configure_common(int cpun, boolean_t boot)
287 {
288         struct cpu *cp;
289         kthread_id_t tp;
290         caddr_t sp;
291         proc_t *procp;
292 #if !defined(__xpv)
293         extern int idle_cpu_prefer_mwait;
294         extern void cpu_idle_mwait();
295 #endif
296         extern void idle();
297         extern void cpu_idle();

299 #ifdef TRAPTRACE
300         trap_trace_ctl_t *ttc = &trap_trace_ctl[cpun];
301 #endif

303         ASSERT(MUTEX_HELD(&cpu_lock));
304         ASSERT(cpun < NCPU && cpu[cpun] == NULL);

306         if (cpu_free_list == NULL) {
307                 cp = kmem_zalloc(sizeof (*cp), KM_SLEEP);
308         } else {
309                 cp = cpu_free_list;
310                 cpu_free_list = cp->cpu_next_free;
311         }

313         cp->cpu_m.mcpu_istamp = cpun << 16;

315         /* Create per CPU specific threads in the process p0. */
```



```
316         procp = &p0;

318         /*
319          * Initialize the dispatcher first.
320          */
321         disp_cpu_init(cp);

323         cpu_vm_data_init(cp);

325         /*
326          * Allocate and initialize the startup thread for this CPU.
327          * Interrupt and process switch stacks get allocated later
328          * when the CPU starts running.
329          */
330         tp = thread_create(NULL, 0, NULL, NULL, 0, procp,
331             TS_STOPPED, maxclsyspri);

333         /*
334          * Set state to TS_ONPROC since this thread will start running
335          * as soon as the CPU comes online.
336          *
337          * All the other fields of the thread structure are setup by
338          * thread_create().
339          */
340         THREAD_ONPROC(tp, cp);
341         tp->t_preempt = 1;
342         tp->t_bound_cpu = cp;
343         tp->t_affinitycnt = 1;
344         tp->t_cpu = cp;
345         tp->t_disp_queue = cp->cpu_disp;

347         /*
348          * Setup thread to start in mp_startup_common.
349          */
350         sp = tp->t_stk;
351         tp->t_sp = (uintptr_t)(sp - MINFRAME);
352 #if defined(__amd64)
353         tp->t_sp -= STACK_ENTRY_ALIGN;          /* fake a call */
354 #endif
355         /*
356          * Setup thread start entry point for boot or hotplug.
357          */
358         if (boot) {
359                 tp->t_pc = (uintptr_t)mp_startup_boot;
360         } else {
361                 tp->t_pc = (uintptr_t)mp_startup_hotplug;
362         }

364         cp->cpu_id = cpun;
365         cp->cpu_self = cp;
366         cp->cpu_thread = tp;
367         cp->cpu_lwp = NULL;
368         cp->cpu_dispthread = tp;
369         cp->cpu_dispatch_pri = DISP_PRIO(tp);

371         /*
372          * cpu_base_spl must be set explicitly here to prevent any blocking
373          * operations in mp_startup_common from causing the spl of the cpu
374          * to drop to 0 (allowing device interrupts before we're ready) in
375          * resume().
376          * cpu_base_spl MUST remain at LOCK_LEVEL until the cpu is CPU_READY.
377          * As an extra bit of security on DEBUG kernels, this is enforced with
378          * an assertion in mp_startup_common() -- before cpu_base_spl is set
379          * to its proper value.
380          */
381         cp->cpu_base_spl = ipltospl(LOCK_LEVEL);
```

```
383         /*
384          * Now, initialize per-CPU idle thread for this CPU.
385          */
386         tp = thread_create(NULL, PAGESIZE, idle, NULL, 0, procp, TS_ONPROC, -1);

388         cp->cpu_idle_thread = tp;

390         tp->t_preempt = 1;
391         tp->t_bound_cpu = cp;
392         tp->t_affinitycnt = 1;
393         tp->t_cpu = cp;
394         tp->t_disp_queue = cp->cpu_disp;

396         /*
397          * Bootstrap the CPU's PG data
398          */
399         pg_cpu_bootstrap(cp);

401         /*
402          * Perform CPC initialization on the new CPU.
403          */
404         kcpc_hw_init(cp);

406         /*
407          * Allocate virtual addresses for cpu_caddr1 and cpu_caddr2
408          * for each CPU.
409          */
410         setup_vaddr_for_ppcopy(cp);

412         /*
413          * Allocate page for new GDT and initialize from current GDT.
414          */
415 #if !defined(__lint)
416         ASSERT((sizeof (*cp->cpu_gdt) * NGDT) <= PAGESIZE);
417 #endif
418         cp->cpu_gdt = kmem_zalloc(PAGESIZE, KM_SLEEP);
419         bcopy(CPU->cpu_gdt, cp->cpu_gdt, (sizeof (*cp->cpu_gdt) * NGDT));

421 #if defined(__i386)
422         /*
423          * setup kernel %gs.
424          */
425         set_usegd(&cp->cpu_gdt[GDT_GS], cp, sizeof (struct cpu) -1, SDT_MEMRWA,
426             SEL_KPL, 0, 1);
427 #endif

429         /*
430          * Allocate pages for the CPU LDT.
421          * If we have more than one node, each cpu gets a copy of IDT
422          * local to its node. If this is a Pentium box, we use cpu 0's
423          * IDT. cpu 0's IDT has been made read-only to workaround the
424          * cmpxchgl register bug
431          */
432         cp->cpu_m.mcpu_ldt = kmem_zalloc(LDT_CPU_SIZE, KM_SLEEP);
433         cp->cpu_m.mcpu_ldt_len = 0;

435         /*
436          * Allocate a per-CPU IDT and initialize the new IDT to the currently
437          * runing CPU.
438          */
426         if (system_hardware.hd_nodes && x86_type != X86_TYPE_P5) {
439 #if !defined(__lint)
440         ASSERT((sizeof (*CPU->cpu_idt) * NIDT) <= PAGESIZE);
441 #endif
442         cp->cpu_idt = kmem_alloc(PAGESIZE, KM_SLEEP);
```



```
430                 cp->cpu_idt = kmem_zalloc(PAGESIZE, KM_SLEEP);
443         bcopy(CPU->cpu_idt, cp->cpu_idt, PAGESIZE);
432         } else {
433                 cp->cpu_idt = CPU->cpu_idt;
434         }

445         /*
446          * alloc space for cpuid info
447          */
448         cpuid_alloc_space(cp);
449 #if !defined(__xpv)
450         if (is_x86_feature(x86_featureset, X86FSET_MWAIT) &&
451             idle_cpu_prefer_mwait) {
452                 cp->cpu_m.mcpu_mwait = cpuid_mwait_alloc(cp);
453                 cp->cpu_m.mcpu_idle_cpu = cpu_idle_mwait;
454         } else
455 #endif
456                 cp->cpu_m.mcpu_idle_cpu = cpu_idle;

458         init_cpu_info(cp);

460 #if !defined(__xpv)
461         init_cpu_id_gdt(cp);
462 #endif

464         /*
465          * alloc space for ucode_info
466          */
467         ucode_alloc_space(cp);
468         xc_init_cpu(cp);
469         hat_cpu_online(cp);

471 #ifdef TRAPTRACE
472         /*
473          * If this is a TRAPTRACE kernel, allocate TRAPTRACE buffers
474          */
475         ttc->ttc_first = (uintptr_t)kmem_zalloc(trap_trace_bufsize, KM_SLEEP);
476         ttc->ttc_next = ttc->ttc_first;
477         ttc->ttc_limit = ttc->ttc_first + trap_trace_bufsize;
478 #endif

480         /*
481          * Record that we have another CPU.
482          */
483         /*
484          * Initialize the interrupt threads for this CPU
485          */
486         cpu_intr_alloc(cp, NINTR_THREADS);

488         cp->cpu_flags = CPU_OFFLINE | CPU_QUIESCED | CPU_POWEROFF;
489         cpu_set_state(cp);

491         /*
492          * Add CPU to list of available CPUs.  It'll be on the active list
493          * after mp_startup_common().
494          */
495         cpu_add_unit(cp);

497         return (cp);
498 }

500 /*
501  * Undo what was done in mp_cpu_configure_common
502  */
503 static void
504 mp_cpu_unconfigure_common(struct cpu *cp, int error)
```

```
 505 {
 506         ASSERT(MUTEX_HELD(&cpu_lock));
 507
 508         /*
 509          * Remove the CPU from the list of available CPUs.
 510          */
 511         cpu_del_unit(cp->cpu_id);
 512
 513         if (error == ETIMEDOUT) {
 514                 /*
 515                  * The cpu was started, but never *seemed* to run any
 516                  * code in the kernel; it's probably off spinning in its
 517                  * own private world, though with potential references to
 518                  * our kmem-allocated IDTs and GDTs (for example).
 519                  *
 520                  * Worse still, it may actually wake up some time later,
 521                  * so rather than guess what it might or might not do, we
 522                  * leave the fundamental data structures intact.
 523                  */
 524                 cp->cpu_flags = 0;
 525                 return;
 526         }
 527
 528         /*
 529          * At this point, the only threads bound to this CPU should
 530          * special per-cpu threads: it's idle thread, it's pause threads,
 531          * and it's interrupt threads.  Clean these up.
 532          */
 533         cpu_destroy_bound_threads(cp);
 534         cp->cpu_idle_thread = NULL;
 535
 536         /*
 537          * Free the interrupt stack.
 538          */
 539         segkp_release(segkp,
 540             cp->cpu_intr_stack - (INTR_STACK_SIZE - SA(MINFRAME)));
 541         cp->cpu_intr_stack = NULL;
 542
 543 #ifdef TRAPTRACE
 544         /*
 545          * Discard the trap trace buffer
 546          */
 547         {
 548                 trap_trace_ctl_t *ttc = &trap_trace_ctl[cp->cpu_id];
 549
 550                 kmem_free((void *)ttc->ttc_first, trap_trace_bufsize);
 551                 ttc->ttc_first = NULL;
 552         }
 553 #endif
 554
 555         hat_cpu_offline(cp);
 556
 557         ucode_free_space(cp);
 558
 559         /* Free CPU ID string and brand string. */
 560         if (cp->cpu_idstr) {
 561                 kmem_free(cp->cpu_idstr, CPU_IDSTRLEN);
 562                 cp->cpu_idstr = NULL;
 563         }
 564         if (cp->cpu_brandstr) {
 565                 kmem_free(cp->cpu_brandstr, CPU_IDSTRLEN);
 566                 cp->cpu_brandstr = NULL;
 567         }
 568
 569 #if !defined(__xpv)
 570         if (cp->cpu_m.mcpu_mwait != NULL) {
```



```
 571                 cpuid_mwait_free(cp);
 572                 cp->cpu_m.mcpu_mwait = NULL;
 573         }
 574 #endif
 575         cpuid_free_space(cp);
 576
 577         if (cp->cpu_idt != CPU->cpu_idt)
 578                 kmem_free(cp->cpu_idt, PAGESIZE);
 579         cp->cpu_idt = NULL;
 580
 581         kmem_free(cp->cpu_m.mcpu_ldt, LDT_CPU_SIZE);
 582         cp->cpu_m.mcpu_ldt = NULL;
 583         cp->cpu_m.mcpu_ldt_len = 0;
 584
 585         kmem_free(cp->cpu_gdt, PAGESIZE);
 586         cp->cpu_gdt = NULL;
 587
 588         if (cp->cpu_supp_freqs != NULL) {
 589                 size_t len = strlen(cp->cpu_supp_freqs) + 1;
 590                 kmem_free(cp->cpu_supp_freqs, len);
 591                 cp->cpu_supp_freqs = NULL;
 592         }
 593
 594         teardown_vaddr_for_ppcopy(cp);
 595
 596         kcpc_hw_fini(cp);
 597
 598         cp->cpu_dispthread = NULL;
 599         cp->cpu_thread = NULL;  /* discarded by cpu_destroy_bound_threads() */
 600
 601         cpu_vm_data_destroy(cp);
 602
 603         xc_fini_cpu(cp);
 604         disp_cpu_fini(cp);
 605
 606         ASSERT(cp != CPU0);
 607         bzero(cp, sizeof (*cp));
 608         cp->cpu_next_free = cpu_free_list;
 609         cpu_free_list = cp;
 610 }
_____unchanged_portion_omitted_

1663 /*
1664  * Startup function for 'other' CPUs (besides boot cpu).
1665  * Called from real_mode_start.
1666  *
1667  * WARNING: until CPU_READY is set, mp_startup_common and routines called by
1668  * mp_startup_common should not call routines (e.g. kmem_free) that could call
1669  * hat_unload which requires CPU_READY to be set.
1670  */
1671 static void
1672 mp_startup_common(boolean_t boot)
1673 {
1674         cpu_t *cp = CPU;
1675         uchar_t new_x86_featureset[BT_SIZEOFMAP(NUM_X86_FEATURES)];
1676         extern void cpu_event_init_cpu(cpu_t *);
1677
1678         /*
1679          * We need to get TSC on this proc synced (i.e., any delta
1680          * from cpu0 accounted for) as soon as we can, because many
1681          * many things use gethrtime/pc_gethrestime, including
1682          * interrupts, cmn_err, etc.  Before we can do that, we want to
1683          * clear TSC if we're on a buggy Sandy/Ivy Bridge CPU, so do that
1684          * right away.
1685          */
1686         bzero(new_x86_featureset, BT_SIZEOFMAP(NUM_X86_FEATURES));
```

```
1687         cpuid_pass1(cp, new_x86_featureset);

1689         if (boot && get_hwenv() == HW_NATIVE &&
1690             cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
1691             cpuid_getfamily(CPU) == 6 &&
1692             (cpuid_getmodel(CPU) == 0x2d || cpuid_getmodel(CPU) == 0x3e) &&
1693             is_x86_feature(new_x86_featureset, X86FSET_TSC)) {
1694                 (void) wrmsr(REG_TSC, 0UL);
1695         }

1697         /* Let the control CPU continue into tsc_sync_master() */
1698         mp_startup_signal(&procset_slave, cp->cpu_id);

1700 #ifndef __xpv
1701         if (tsc_gethrtime_enable)
1702                 tsc_sync_slave();
1703 #endif

1705         /*
1706          * Once this was done from assembly, but it's safer here; if
1707          * it blocks, we need to be able to swtch() to and from, and
1708          * since we get here by calling t_pc, we need to do that call
1709          * before swtch() overwrites it.
1710          */
1711         (void) (*ap_mlsetup)();

1713 #ifndef __xpv
1714         /*
1715          * Program this cpu's PAT
1716          */
1717         pat_sync();
1718 #endif

1720         /*
1721          * Set up TSC_AUX to contain the cpuid for this processor
1722          * for the rdtscp instruction.
1723          */
1724         if (is_x86_feature(x86_featureset, X86FSET_TSCP))
1725                 (void) wrmsr(MSR_AMD_TSCAUX, cp->cpu_id);

1727         /*
1728          * Initialize this CPU's syscall handlers
1729          */
1730         init_cpu_syscall(cp);

1732         /*
1733          * Enable interrupts with spl set to LOCK_LEVEL. LOCK_LEVEL is the
1734          * highest level at which a routine is permitted to block on
1735          * an adaptive mutex (allows for cpu poke interrupt in case
1736          * the cpu is blocked on a mutex and halts). Setting LOCK_LEVEL blocks
1737          * device interrupts that may end up in the hat layer issuing cross
1738          * calls before CPU_READY is set.
1739          */
1740         splx(ipltospl(LOCK_LEVEL));
1741         sti();

1743         /*
1744          * Do a sanity check to make sure this new CPU is a sane thing
1745          * to add to the collection of processors running this system.
1746          *
1747          * XXX  Clearly this needs to get more sophisticated, if x86
1748          * systems start to get built out of heterogenous CPUs; as is
1749          * likely to happen once the number of processors in a configuration
1750          * gets large enough.
1751          */
1752         if (compare_x86_featureset(x86_featureset, new_x86_featureset) ==
```



```
1753             B_FALSE) {
1754                 cmn_err(CE_CONT, "cpu%d: featureset\n", cp->cpu_id);
1755                 print_x86_featureset(new_x86_featureset);
1756                 cmn_err(CE_WARN, "cpu%d feature mismatch", cp->cpu_id);
1757         }

1759         /*
1760          * There exists a small subset of systems which expose differing
1761          * MWAIT/MONITOR support between CPUs.  If MWAIT support is absent from
1762          * the boot CPU, but is found on a later CPU, the system continues to
1763          * operate as if no MWAIT support is available.
1764          *
1765          * The reverse case, where MWAIT is available on the boot CPU but not
1766          * on a subsequently initialized CPU, is not presently allowed and will
1767          * result in a panic.
1768          */
1769         if (is_x86_feature(x86_featureset, X86FSET_MWAIT) !=
1770             is_x86_feature(new_x86_featureset, X86FSET_MWAIT)) {
1771                 if (!is_x86_feature(x86_featureset, X86FSET_MWAIT)) {
1772                         remove_x86_feature(new_x86_featureset, X86FSET_MWAIT);
1773                 } else {
1774                         panic("unsupported mixed cpu mwait support detected");
1775                 }
1776         }

1778         /*
1779          * We could be more sophisticated here, and just mark the CPU
1780          * as "faulted" but at this point we'll opt for the easier
1781          * answer of dying horribly.  Provided the boot cpu is ok,
1782          * the system can be recovered by booting with use_mp set to zero.
1783          */
1784         if (workaround_errata(cp) != 0)
1785                 panic("critical workaround(s) missing for cpu%d", cp->cpu_id);

1787         /*
1788          * We can touch cpu_flags here without acquiring the cpu_lock here
1789          * because the cpu_lock is held by the control CPU which is running
1790          * mp_start_cpu_common().
1791          * Need to clear CPU_QUIESCED flag before calling any function which
1792          * may cause thread context switching, such as kmem_alloc() etc.
1793          * The idle thread checks for CPU_QUIESCED flag and loops for ever if
1794          * it's set. So the startup thread may have no chance to switch back
1795          * again if it's switched away with CPU_QUIESCED set.
1796          */
1797         cp->cpu_flags &= ~(CPU_POWEROFF | CPU_QUIESCED);

1799         enable_pcid();

1801         /*
1802          * Setup this processor for XSAVE.
1803          */
1804         if (fp_save_mech == FP_XSAVE) {
1805                 xsave_setup_msr(cp);
1806         }

1808         cpuid_pass2(cp);
1809         cpuid_pass3(cp);
1810         cpuid_pass4(cp, NULL);

1812         /*
1813          * Correct cpu_idstr and cpu_brandstr on target CPU after
1814          * cpuid_pass1() is done.
1815          */
1816         (void) cpuid_getidstr(cp, cp->cpu_idstr, CPU_IDSTRLEN);
1817         (void) cpuid_getbrandstr(cp, cp->cpu_brandstr, CPU_IDSTRLEN);
```

```
1819         cp->cpu_flags |= CPU_RUNNING | CPU_READY | CPU_EXISTS;

1821         post_startup_cpu_fixups();

1823         cpu_event_init_cpu(cp);

1825         /*
1826          * Enable preemption here so that contention for any locks acquired
1827          * later in mp_startup_common may be preempted if the thread owning
1828          * those locks is continuously executing on other CPUs (for example,
1829          * this CPU must be preemptible to allow other CPUs to pause it during
1830          * their startup phases).  It's safe to enable preemption here because
1831          * the CPU state is pretty-much fully constructed.
1832          */
1833         curthread->t_preempt = 0;

1835         /* The base spl should still be at LOCK_LEVEL here */
1836         ASSERT(cp->cpu_base_spl == ipltospl(LOCK_LEVEL));
1837         set_base_spl();         /* Restore the spl to its proper value */

1839         pghw_physid_create(cp);
1840         /*
1841          * Delegate initialization tasks, which need to access the cpu_lock,
1842          * to mp_start_cpu_common() because we can't acquire the cpu_lock here
1843          * during CPU DR operations.
1844          */
1845         mp_startup_signal(&procset_slave, cp->cpu_id);
1846         mp_startup_wait(&procset_master, cp->cpu_id);
1847         pg_cmt_cpu_startup(cp);

1849         if (boot) {
1850                 mutex_enter(&cpu_lock);
1851                 cp->cpu_flags &= ~CPU_OFFLINE;
1852                 cpu_enable_intr(cp);
1853                 cpu_add_active(cp);
1854                 mutex_exit(&cpu_lock);
1855         }

1857         /* Enable interrupts */
1858         (void) spl0();

1860         /*
1861          * Fill out cpu_ucode_info.  Update microcode if necessary.
1862          */
1863         ucode_check(cp);

1865 #ifndef __xpv
1866         {
1867                 /*
1868                  * Set up the CPU module for this CPU.  This can't be done
1869                  * before this CPU is made CPU_READY, because we may (in
1870                  * heterogeneous systems) need to go load another CPU module.
1871                  * The act of attempting to load a module may trigger a
1872                  * cross-call, which will ASSERT unless this cpu is CPU_READY.
1873                  */
1874                 cmi_hdl_t hdl;

1876                 if ((hdl = cmi_init(CMI_HDL_NATIVE, cmi_ntv_hwchipid(CPU),
1877                     cmi_ntv_hwcoreid(CPU), cmi_ntv_hwstrandid(CPU))) != NULL) {
1878                         if (is_x86_feature(x86_featureset, X86FSET_MCA))
1879                                 cmi_mca_init(hdl);
1880                         cp->cpu_m.mcpu_cmi_hdl = hdl;
1881                 }
1882         }
1883 #endif /* __xpv */
```



```
1885         if (boothowto & RB_DEBUG)
1886                 kdi_cpu_init();

1888         /*
1889          * Setting the bit in cpu_ready_set must be the last operation in
1890          * processor initialization; the boot CPU will continue to boot once
1891          * it sees this bit set for all active CPUs.
1892          */
1893         CPUSET_ATOMIC_ADD(cpu_ready_set, cp->cpu_id);

1895         (void) mach_cpu_create_device_node(cp, NULL);

1897         cmn_err(CE_CONT, "?cpu%d: %s\n", cp->cpu_id, cp->cpu_idstr);
1898         cmn_err(CE_CONT, "?cpu%d: %s\n", cp->cpu_id, cp->cpu_brandstr);
1899         cmn_err(CE_CONT, "?cpu%d initialization complete - online\n",
1900             cp->cpu_id);

1902         /*
1903          * Now we are done with the startup thread, so free it up.
1904          */
1905         thread_exit();
1906         panic("mp_startup: cannot return");
1907         /*NOTREACHED*/
1908 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   88262 Fri Apr  6 17:25:03 2018
new/usr/src/uts/i86pc/os/startup.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 DEY Storage Systems, Inc.  All rights reserved.
 * Copyright 2017 Nexenta Systems, Inc.
 * Copyright (c) 2018 Joyent, Inc.
 * Copyright 2015 Joyent, Inc.
 * Copyright (c) 2015 by Delphix. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */

#include <sys/types.h>
#include <sys/t_lock.h>
#include <sys/param.h>
#include <sys/sysmacros.h>
#include <sys/signal.h>
#include <sys/systm.h>
#include <sys/user.h>
#include <sys/mman.h>
#include <sys/vm.h>
#include <sys/conf.h>
#include <sys/avintr.h>
#include <sys/autoconf.h>
#include <sys/disp.h>
#include <sys/class.h>
#include <sys/bitmap.h>

#include <sys/privregs.h>

#include <sys/proc.h>
#include <sys/buf.h>
#include <sys/kmem.h>
#include <sys/mem.h>
#include <sys/kstat.h>

#include <sys/reboot.h>
```



```
#include <sys/cred.h>
#include <sys/vnode.h>
#include <sys/file.h>

#include <sys/procfs.h>

#include <sys/vfs.h>
#include <sys/cmn_err.h>
#include <sys/utsname.h>
#include <sys/debug.h>
#include <sys/kdi.h>

#include <sys/dumphdr.h>
#include <sys/bootconf.h>
#include <sys/memlist_plat.h>
#include <sys/varargs.h>
#include <sys/promif.h>
#include <sys/modctl.h>

#include <sys/sunddi.h>
#include <sys/sunndi.h>
#include <sys/ndi_impldefs.h>
#include <sys/ddidmareq.h>
#include <sys/psw.h>
#include <sys/regset.h>
#include <sys/clock.h>
#include <sys/pte.h>
#include <sys/tss.h>
#include <sys/stack.h>
#include <sys/trap.h>
#include <sys/fp.h>
#include <vm/kboot_mmu.h>
#include <vm/anon.h>
#include <vm/as.h>
#include <vm/page.h>
#include <vm/seg.h>
#include <vm/seg_dev.h>
#include <vm/seg_kmem.h>
#include <vm/seg_kpm.h>
#include <vm/seg_map.h>
#include <vm/seg_vn.h>
#include <vm/seg_kp.h>
#include <sys/memnode.h>
#include <vm/vm_dep.h>
#include <sys/thread.h>
#include <sys/sysconf.h>
#include <sys/vm_machparam.h>
#include <sys/archsystm.h>
#include <sys/machsystm.h>
#include <vm/hat.h>
#include <vm/hat_i86.h>
#include <sys/pmem.h>
#include <sys/smp_impldefs.h>
#include <sys/x86_archext.h>
#include <sys/cpuvar.h>
#include <sys/segments.h>
#include <sys/clconf.h>
#include <sys/kobj.h>
#include <sys/kobj_lex.h>
#include <sys/cpc_impl.h>
#include <sys/cpu_module.h>
#include <sys/smbios.h>
#include <sys/debug_info.h>
#include <sys/bootinfo.h>
#include <sys/ddi_periodic.h>
```

```
125 #include <sys/systeminfo.h>
126 #include <sys/multiboot.h>
127 #include <sys/ramdisk.h>

129 #ifdef  __xpv

131 #include <sys/hypervisor.h>
132 #include <sys/xen_mmu.h>
133 #include <sys/evtchn_impl.h>
134 #include <sys/gnttab.h>
135 #include <sys/xpv_panic.h>
136 #include <xen/sys/xenbus_comms.h>
137 #include <xen/public/physdev.h>

139 extern void xen_late_startup(void);

141 struct xen_evt_data cpu0_evt_data;

143 #else   /* __xpv */
144 #include <sys/memlist_impl.h>

146 extern void mem_config_init(void);
147 #endif /* __xpv */

149 extern void progressbar_init(void);
150 extern void brand_init(void);
151 extern void pcf_init(void);
152 extern void pg_init(void);
153 extern void ssp_init(void);

155 extern int size_pse_array(pgcnt_t, int);

157 #if defined(_SOFT_HOSTID)

159 #include <sys/rtc.h>

161 static int32_t set_soft_hostid(void);
162 static char hostid_file[] = "/etc/hostid";

164 #endif

166 void *gfx_devinfo_list;

168 #if defined(__amd64) && !defined(__xpv)
169 extern void immu_startup(void);
170 #endif

172 /*
173  * XXX make declaration below "static" when drivers no longer use this
174  * interface.
175  */
176 extern caddr_t p0_va;   /* Virtual address for accessing physical page 0 */

178 /*
179  * segkp
180  */
181 extern int segkp_fromheap;

183 static void kvm_init(void);
184 static void startup_init(void);
185 static void startup_memlist(void);
186 static void startup_kmem(void);
187 static void startup_modules(void);
188 static void startup_vm(void);
189 static void startup_end(void);
190 static void layout_kernel_va(void);
```



```
192 /*
193  * Declare these as initialized data so we can patch them.
194  */
195 #ifdef __i386

197 /*
198  * Due to virtual address space limitations running in 32 bit mode, restrict
199  * the amount of physical memory configured to a max of PHYSMEM pages (16g).
200  *
201  * If the physical max memory size of 64g were allowed to be configured, the
202  * size of user virtual address space will be less than 1g. A limited user
203  * address space greatly reduces the range of applications that can run.
204  *
205  * If more physical memory than PHYSMEM is required, users should preferably
206  * run in 64 bit mode which has far looser virtual address space limitations.
207  *
208  * If 64 bit mode is not available (as in IA32) and/or more physical memory
209  * than PHYSMEM is required in 32 bit mode, physmem can be set to the desired
210  * value or to 0 (to configure all available memory) via eeprom(1M). kernelbase
211  * should also be carefully tuned to balance out the need of the user
212  * application while minimizing the risk of kernel heap exhaustion due to
213  * kernelbase being set too high.
214  */
215 #define PHYSMEM 0x400000

217 #else /* __amd64 */

219 /*
220  * For now we can handle memory with physical addresses up to about
221  * 64 Terabytes. This keeps the kernel above the VA hole, leaving roughly
222  * half the VA space for seg_kpm. When systems get bigger than 64TB this
223  * code will need revisiting. There is an implicit assumption that there
224  * are no *huge* holes in the physical address space too.
225  */
226 #define TERABYTE                (1ul << 40)
227 #define PHYSMEM_MAX64           mmu_btop(64 * TERABYTE)
228 #define PHYSMEM                 PHYSMEM_MAX64
229 #define AMD64_VA_HOLE_END       0xFFFF800000000000ul

231 #endif /* __amd64 */

233 pgcnt_t physmem = PHYSMEM;
234 pgcnt_t obp_pages;      /* Memory used by PROM for its text and data */

236 char *kobj_file_buf;
237 int kobj_file_bufsize;  /* set in /etc/system */

239 /* Global variables for MP support. Used in mp_startup */
240 caddr_t rm_platter_va = 0;
241 uint32_t rm_platter_pa;

243 int     auto_lpg_disable = 1;

245 /*
246  * Some CPUs have holes in the middle of the 64-bit virtual address range.
247  */
248 uintptr_t hole_start, hole_end;

250 /*
251  * kpm mapping window
252  */
253 caddr_t kpm_vbase;
254 size_t  kpm_size;
255 static int kpm_desired;
256 #ifdef __amd64
```

```
257 static uintptr_t segkpm_base = (uintptr_t)SEGKPM_BASE;
258 #endif

260 /*
261  * Configuration parameters set at boot time.
262  */

264 caddr_t econtig;               /* end of first block of contiguous kernel */

266 struct bootops          *bootops = 0;   /* passed in from boot */
267 struct bootops          **bootopsp;
268 struct boot_syscalls    *sysp;          /* passed in from boot */

270 char bootblock_fstype[16];

272 char kern_bootargs[OBP_MAXPATHLEN];
273 char kern_bootfile[OBP_MAXPATHLEN];

275 /*
276  * ZFS zio segment.  This allows us to exclude large portions of ZFS data that
277  * gets cached in kmem caches on the heap.  If this is set to zero, we allocate
278  * zio buffers from their own segment, otherwise they are allocated from the
279  * heap.  The optimization of allocating zio buffers from their own segment is
280  * only valid on 64-bit kernels.
281  */
282 #if defined(__amd64)
283 int segzio_fromheap = 0;
284 #else
285 int segzio_fromheap = 1;
286 #endif

288 /*
289  * Give folks an escape hatch for disabling SMAP via kmdb. Doesn't work
290  * post-boot.
291  */
292 int disable_smap = 0;

294 /*
295  * new memory fragmentations are possible in startup() due to BOP_ALLOCs. this
296  * depends on number of BOP_ALLOC calls made and requested size, memory size
297  * combination and whether boot.bin memory needs to be freed.
298  */
299 #define POSS_NEW_FRAGMENTS      12

301 /*
302  * VM data structures
303  */
304 long page_hashsz;               /* Size of page hash table (power of two) */
305 unsigned int page_hashsz_shift; /* log2(page_hashsz) */
306 struct page *pp_base;           /* Base of initial system page struct array */
307 struct page **page_hash;        /* Page hash table */
308 pad_mutex_t *pse_mutex;         /* Locks protecting pp->p_selock */
309 size_t pse_table_size;          /* Number of mutexes in pse_mutex[] */
310 int pse_shift;                  /* log2(pse_table_size) */
311 struct seg ktextseg;            /* Segment used for kernel executable image */
312 struct seg kvalloc;             /* Segment used for "valloc" mapping */
313 struct seg kpseg;               /* Segment used for pageable kernel virt mem */
314 struct seg kmapseg;             /* Segment used for generic kernel mappings */
315 struct seg kdebugseg;           /* Segment used for the kernel debugger */

317 struct seg *segkmap = &kmapseg; /* Kernel generic mapping segment */
318 static struct seg *segmap = &kmapseg;   /* easier to use name for in here */

320 struct seg *segkp = &kpseg;     /* Pageable kernel virtual memory segment */

322 #if defined(__amd64)
```



```
323 struct seg kvseg_core;          /* Segment used for the core heap */
324 struct seg kpmseg;              /* Segment used for physical mapping */
325 struct seg *segkpm = &kpmseg;   /* 64bit kernel physical mapping segment */
326 #else
327 struct seg *segkpm = NULL;      /* Unused on IA32 */
328 #endif

330 caddr_t segkp_base;             /* Base address of segkp */
331 caddr_t segzio_base;            /* Base address of segzio */
332 #if defined(__amd64)
333 pgcnt_t segkpsize = btop(SEGKPDEFSIZE); /* size of segkp segment in pages */
334 #else
335 pgcnt_t segkpsize = 0;
336 #endif
337 pgcnt_t segziosize = 0;         /* size of zio segment in pages */

339 /*
340  * A static DR page_t VA map is reserved that can map the page structures
341  * for a domain's entire RA space. The pages that back this space are
342  * dynamically allocated and need not be physically contiguous.  The DR
343  * map size is derived from KPM size.
344  * This mechanism isn't used by x86 yet, so just stubs here.
345  */
346 int ppvm_enable = 0;            /* Static virtual map for page structs */
347 page_t *ppvm_base = NULL;       /* Base of page struct map */
348 pgcnt_t ppvm_size = 0;          /* Size of page struct map */

350 /*
351  * VA range available to the debugger
352  */
353 const caddr_t kdi_segdebugbase = (const caddr_t)SEGDEBUGBASE;
354 const size_t kdi_segdebugsize = SEGDEBUGSIZE;

356 struct memseg *memseg_base;
357 struct vnode unused_pages_vp;

359 #define FOURGB  0x100000000LL

361 struct memlist *memlist;

363 caddr_t s_text;         /* start of kernel text segment */
364 caddr_t e_text;         /* end of kernel text segment */
365 caddr_t s_data;         /* start of kernel data segment */
366 caddr_t e_data;         /* end of kernel data segment */
367 caddr_t modtext;        /* start of loadable module text reserved */
368 caddr_t e_modtext;      /* end of loadable module text reserved */
369 caddr_t moddata;        /* start of loadable module data reserved */
370 caddr_t e_moddata;      /* end of loadable module data reserved */

372 struct memlist *phys_install;   /* Total installed physical memory */
373 struct memlist *phys_avail;     /* Total available physical memory */
374 struct memlist *bios_rsvd;      /* Bios reserved memory */

376 /*
377  * kphysm_init returns the number of pages that were processed
378  */
379 static pgcnt_t kphysm_init(page_t *, pgcnt_t);

381 #define IO_PROP_SIZE    64      /* device property size */

383 /*
384  * a couple useful roundup macros
385  */
386 #define ROUND_UP_PAGE(x)        \
387         ((uintptr_t)P2ROUNDUP((uintptr_t)(x), (uintptr_t)MMU_PAGESIZE))
388 #define ROUND_UP_LPAGE(x)       \
```

```
389         ((uintptr_t)P2ROUNDUP((uintptr_t)(x), mmu.level_size[1]))
390 #define ROUND_UP_4MEG(x)        \
391         ((uintptr_t)P2ROUNDUP((uintptr_t)(x), (uintptr_t)FOUR_MEG))
392 #define ROUND_UP_TOPLEVEL(x)    \
393         ((uintptr_t)P2ROUNDUP((uintptr_t)(x), mmu.level_size[mmu.max_level]))

395 /*
396  *      32-bit Kernel's Virtual memory layout.
397  *                +-----------------------+
398  *                |                       |
399  * 0xFFC00000  -  |-----------------------|- ARGSBASE
400  *                |       debugger        |
401  * 0xFF800000  -  |-----------------------|- SEGDEBUGBASE
402  *                |      Kernel Data      |
403  * 0xFEC00000  -  |-----------------------|
404  *                |      Kernel Text      |
405  * 0xFE800000  -  |-----------------------|- KERNEL_TEXT (0xFB400000 on Xen)
406  *                |---       GDT        ---|- GDT page (GDT_VA)
407  *                |---    debug info    ---|- debug info (DEBUG_INFO_VA)
408  *                |                       |
409  *                |   page_t structures   |
410  *                |   memsegs, memlists,  |
411  *                |   page hash, etc.     |
412  * ---            |-----------------------|- ekernelheap, valloc_base (floating)
413  *                |                       |  (segkp is just an arena in the heap)
414  *                |                       |
415  *                |       kvseg           |
416  *                |                       |
417  *                |                       |
418  * ---            |-----------------------|- kernelheap (floating)
419  *                |        Segkmap        |
420  * 0xC3002000  -  |-----------------------|- segmap_start (floating)
421  *                |       Red Zone        |
422  * 0xC3000000  -  |-----------------------|- kernelbase / userlimit (floating)
423  *                |                       |                           ||
424  *                |     Shared objects    |                           \/
425  *                |                       |
426  *                :                       :
427  *                |       user data       |
428  *                |-----------------------|
429  *                |       user text       |
430  * 0x08048000  -  |-----------------------|
431  *                |       user stack      |
432  *                :                       :
433  *                |       invalid         |
434  * 0x00000000     +-----------------------+
435  *
436  *
437  *                64-bit Kernel's Virtual memory layout. (assuming 64 bit app)
438  *                          +-----------------------+
439  *                          |                       |
440  * 0xFFFFFFFF.FFC00000      |-----------------------|- ARGSBASE
441  *                          |       debugger (?)    |
442  * 0xFFFFFFFF.FF800000      |-----------------------|- SEGDEBUGBASE
443  *                          |      unused           |
444  *                          +-----------------------+
445  *                          |      Kernel Data      |
446  * 0xFFFFFFFF.FBC00000      |-----------------------|
447  *                          |      Kernel Text      |
448  * 0xFFFFFFFF.FB800000      |-----------------------|- KERNEL_TEXT
449  *                          |---    debug info    ---|- debug info (DEBUG_INFO_VA)
450  *                          |---       GDT        ---|- GDT page (GDT_VA)
451  *                          |---       IDT        ---|- IDT page (IDT_VA)
452  *                          |---       LDT        ---|- LDT pages (LDT_VA)
450  *                          |---    debug info    ---|- debug info (DEBUG_INFO_VA)
453  *                          |                       |
```



```
454  *                          |      Core heap        | (used for loadable modules)
455  * 0xFFFFFFFF.C0000000      |-----------------------|- core_base / ekernelheap
456  *                          |        Kernel         |
457  *                          |         heap          |
458  * 0xFFFFFXXX.XXX00000      |-----------------------|- kernelheap (floating)
459  *                          |        segmap         |
460  * 0xFFFFFXXX.XXX00000      |-----------------------|- segmap_start (floating)
461  *                          |    device mappings    |
462  * 0xFFFFFXXX.XXX00000      |-----------------------|- toxic_addr (floating)
463  *                          |         segzio        |
464  * 0xFFFFFXXX.XXX00000      |-----------------------|- segzio_base (floating)
465  *                          |         segkp         |
466  * ---                      |-----------------------|- segkp_base (floating)
467  *                          |   page_t structures   |  valloc_base + valloc_sz
468  *                          |   memsegs, memlists,  |
469  *                          |   page hash, etc.     |
470  * 0xFFFFFF00.00000000      |-----------------------|- valloc_base (lower if >256GB)
471  *                          |        segkpm         |
472  * 0xFFFFFE00.00000000      |-----------------------|
473  *                          |       Red Zone        |
474  * 0xFFFFFD80.00000000      |-----------------------|- KERNELBASE (lower if >256GB)
475  *                          |     User stack        |- User space memory
476  *                          |                       |
477  *                          | shared objects, etc   |       (grows downwards)
478  *                          :                       :
479  *                          |                       |
480  * 0xFFFF8000.00000000      |-----------------------|
481  *                          |                       |
482  *                          | VA Hole / unused      |
483  *                          |                       |
484  * 0x00008000.00000000      |-----------------------|
485  *                          |                       |
486  *                          |                       |
487  *                          :                       :
488  *                          |       user heap       |       (grows upwards)
489  *                          |                       |
490  *                          |       user data       |
491  *                          |-----------------------|
492  *                          |       user text       |
493  * 0x00000000.04000000      |-----------------------|
494  *                          |       invalid         |
495  * 0x00000000.00000000      +-----------------------+
496  *
497  * A 32 bit app on the 64 bit kernel sees the same layout as on the 32 bit
498  * kernel, except that userlimit is raised to 0xfe000000
499  *
500  * Floating values:
501  *
502  * valloc_base: start of the kernel's memory management/tracking data
503  * structures.  This region contains page_t structures for
504  * physical memory, memsegs, memlists, and the page hash.
505  *
506  * core_base: start of the kernel's "core" heap area on 64-bit systems.
507  * This area is intended to be used for global data as well as for module
508  * text/data that does not fit into the nucleus pages.  The core heap is
509  * restricted to a 2GB range, allowing every address within it to be
510  * accessed using rip-relative addressing
511  *
512  * ekernelheap: end of kernelheap and start of segmap.
513  *
514  * kernelheap: start of kernel heap.  On 32-bit systems, this starts right
515  * above a red zone that separates the user's address space from the
516  * kernel's.  On 64-bit systems, it sits above segkp and segkpm.
517  *
518  * segmap_start: start of segmap. The length of segmap can be modified
519  * through eeprom. The default length is 16MB on 32-bit systems and 64MB
```

```
 520  * on 64-bit systems.
 521  *
 522  * kernelbase: On a 32-bit kernel the default value of 0xd4000000 will be
 523  * decreased by 2X the size required for page_t.  This allows the kernel
 524  * heap to grow in size with physical memory.  With sizeof(page_t) == 80
 525  * bytes, the following shows the values of kernelbase and kernel heap
 526  * sizes for different memory configurations (assuming default segmap and
 527  * segkp sizes).
 528  *
 529  *      mem     size for        kernelbase      kernel heap
 530  *      size    page_t's                        size
 531  *      ----    ---------       ----------      -----------
 532  *      1gb     0x01400000      0xd1800000      684MB
 533  *      2gb     0x02800000      0xcf000000      704MB
 534  *      4gb     0x05000000      0xca000000      744MB
 535  *      6gb     0x07800000      0xc5000000      784MB
 536  *      8gb     0x0a000000      0xc0000000      824MB
 537  *      16gb    0x14000000      0xac000000      984MB
 538  *      32gb    0x28000000      0x84000000      1304MB
 539  *      64gb    0x50000000      0x34000000      1944MB (*)
 540  *
 541  * kernelbase is less than the abi minimum of 0xc0000000 for memory
 542  * configurations above 8gb.
 543  *
 544  * (*) support for memory configurations above 32gb will require manual tuning
 545  * of kernelbase to balance out the need of user applications.
 546  */

 548 /* real-time-clock initialization parameters */
 549 extern time_t process_rtc_config_file(void);

 551 uintptr_t       kernelbase;
 552 uintptr_t       postbootkernelbase;     /* not set till boot loader is gone */
 553 uintptr_t       eprom_kernelbase;
 554 size_t          segmapsize;
 555 uintptr_t       segmap_start;
 556 int             segmapfreelists;
 557 pgcnt_t         npages;
 558 pgcnt_t         orig_npages;
 559 size_t          core_size;              /* size of "core" heap */
 560 uintptr_t       core_base;              /* base address of "core" heap */

 562 /*
 563  * List of bootstrap pages. We mark these as allocated in startup.
 564  * release_bootstrap() will free them when we're completely done with
 565  * the bootstrap.
 566  */
 567 static page_t *bootpages;

 569 /*
 570  * boot time pages that have a vnode from the ramdisk will keep that forever.
 571  */
 572 static page_t *rd_pages;

 574 /*
 575  * Lower 64K
 576  */
 577 static page_t *lower_pages = NULL;
 578 static int lower_pages_count = 0;

 580 struct system_hardware system_hardware;

 582 /*
 583  * Enable some debugging messages concerning memory usage...
 584  */
 585 static void
```



```
 586 print_memlist(char *title, struct memlist *mp)
 587 {
 588         prom_printf("MEMLIST: %s:\n", title);
 589         while (mp != NULL)  {
 590                 prom_printf("\tAddress 0x%" PRIx64 ", size 0x%" PRIx64 "\n",
 591                     mp->ml_address, mp->ml_size);
 592                 mp = mp->ml_next;
 593         }
 594 }
```

*unchanged_portion_omitted_*

```
 922 static void
 923 kpm_init()
 924 {
 925         struct segkpm_crargs b;

 927         /*
 928          * These variables were all designed for sfmmu in which segkpm is
 929          * mapped using a single pagesize - either 8KB or 4MB.  On x86, we
 930          * might use 2+ page sizes on a single machine, so none of these
 931          * variables have a single correct value.  They are set up as if we
 932          * always use a 4KB pagesize, which should do no harm.  In the long
 933          * run, we should get rid of KPM's assumption that only a single
 934          * pagesize is used.
 935          */
 936         kpm_pgshft = MMU_PAGESHIFT;
 937         kpm_pgsz =  MMU_PAGESIZE;
 938         kpm_pgoff = MMU_PAGEOFFSET;
 939         kpmp2pshft = 0;
 940         kpmpnpgs = 1;
 941         ASSERT(((uintptr_t)kpm_vbase & (kpm_pgsz - 1)) == 0);

 943         PRM_POINT("about to create segkpm");
 944         rw_enter(&kas.a_lock, RW_WRITER);

 946         if (seg_attach(&kas, kpm_vbase, kpm_size, segkpm) < 0)
 947                 panic("cannot attach segkpm");

 949         b.prot = PROT_READ | PROT_WRITE;
 950         b.nvcolors = 1;

 952         if (segkpm_create(segkpm, (caddr_t)&b) != 0)
 953                 panic("segkpm_create segkpm");

 955         rw_exit(&kas.a_lock);

 957         kpm_enable = 1;

 959         /*
 960          * As the KPM was disabled while setting up the system, go back and fix
 961          * CPU zero's access to its user page table. This is a bit gross, but
 962          * we have a chicken and egg problem otherwise.
 963          */
 964         ASSERT(CPU->cpu_hat_info->hci_user_l3ptes == NULL);
 965         CPU->cpu_hat_info->hci_user_l3ptes =
 966             (x86pte_t *)hat_kpm_mapin_pfn(CPU->cpu_hat_info->hci_user_l3pfn);
 967 }
```

*unchanged_portion_omitted_*

```
1423 /*
1424  * Layout the kernel's part of address space and initialize kmem allocator.
1425  */
1426 static void
1427 startup_kmem(void)
1428 {
1429         extern void page_set_colorequiv_arr(void);
```

```
#if !defined(__xpv)
        extern uint64_t kpti_kbase;
#endif

        PRM_POINT("startup_kmem() starting...");

#if defined(__amd64)
        if (eprom_kernelbase && eprom_kernelbase != KERNELBASE)
                cmn_err(CE_NOTE, "!kernelbase cannot be changed on 64-bit "
                    "systems.");
        kernelbase = segkpm_base - KERNEL_REDZONE_SIZE;
        core_base = (uintptr_t)COREHEAP_BASE;
        core_size = (size_t)MISC_VA_BASE - COREHEAP_BASE;
#else   /* __i386 */
        /*
         * We configure kernelbase based on:
         *
         * 1. user specified kernelbase via eeprom command. Value cannot exceed
         *    KERNELBASE_MAX. we large page align eprom_kernelbase
         *
         * 2. Default to KERNELBASE and adjust to 2X less the size for page_t.
         *    On large memory systems we must lower kernelbase to allow
         *    enough room for page_t's for all of memory.
         *
         * The value set here, might be changed a little later.
         */
        if (eprom_kernelbase) {
                kernelbase = eprom_kernelbase & mmu.level_mask[1];
                if (kernelbase > KERNELBASE_MAX)
                        kernelbase = KERNELBASE_MAX;
        } else {
                kernelbase = (uintptr_t)KERNELBASE;
                kernelbase -= ROUND_UP_4MEG(2 * valloc_sz);
        }
        ASSERT((kernelbase & mmu.level_offset[1]) == 0);
        core_base = valloc_base;
        core_size = 0;
#endif  /* __i386 */

        PRM_DEBUG(core_base);
        PRM_DEBUG(core_size);
        PRM_DEBUG(kernelbase);

#if defined(__i386)
        segkp_fromheap = 1;
#endif  /* __i386 */

        ekernelheap = (char *)core_base;
        PRM_DEBUG(ekernelheap);

        /*
         * Now that we know the real value of kernelbase,
         * update variables that were initialized with a value of
         * KERNELBASE (in common/conf/param.c).
         *
         * XXX  The problem with this sort of hackery is that the
         *      compiler just may feel like putting the const declarations
         *      (in param.c) into the .text section.  Perhaps they should
         *      just be declared as variables there?
         */

        *(uintptr_t *)&_kernelbase = kernelbase;
        *(uintptr_t *)&_userlimit = kernelbase;
#if defined(__amd64)
        *(uintptr_t *)&_userlimit -= KERNELBASE - USERLIMIT;
#if !defined(__xpv)
```



```
        kpti_kbase = kernelbase;
#endif
#else
        *(uintptr_t *)&_userlimit32 = _userlimit;
#endif
        PRM_DEBUG(_kernelbase);
        PRM_DEBUG(_userlimit);
        PRM_DEBUG(_userlimit32);

        /* We have to re-do this now that we've modified _userlimit. */
        mmu_calc_user_slots();

        layout_kernel_va();

#if defined(__i386)
        /*
         * If segmap is too large we can push the bottom of the kernel heap
         * higher than the base.  Or worse, it could exceed the top of the
         * VA space entirely, causing it to wrap around.
         */
        if (kernelheap >= ekernelheap || (uintptr_t)kernelheap < kernelbase)
                panic("too little address space available for kernelheap,"
                    " use eeprom for lower kernelbase or smaller segmapsize");
#endif  /* __i386 */

        /*
         * Initialize the kernel heap. Note 3rd argument must be > 1st.
         */
        kernelheap_init(kernelheap, ekernelheap,
            kernelheap + MMU_PAGESIZE,
            (void *)core_base, (void *)(core_base + core_size));

#if defined(__xpv)
        /*
         * Link pending events struct into cpu struct
         */
        CPU->cpu_m.mcpu_evt_pend = &cpu0_evt_data;
#endif
        /*
         * Initialize kernel memory allocator.
         */
        kmem_init();

        /*
         * Factor in colorequiv to check additional 'equivalent' bins
         */
        page_set_colorequiv_arr();

        /*
         * print this out early so that we know what's going on
         */
        print_x86_featureset(x86_featureset);

        /*
         * Initialize bp_mapin().
         */
        bp_init(MMU_PAGESIZE, HAT_STORECACHING_OK);

        /*
         * orig_npages is non-zero if physmem has been configured for less
         * than the available memory.
         */
        if (orig_npages) {
                cmn_err(CE_WARN, "!%slimiting physmem to 0x%lx of 0x%lx pages",
                    (npages == PHYSMEM ? "Due to virtual address space " : ""),
                    npages, orig_npages);
```

```
1562         }
1563 #if defined(__i386)
1564         if (eprom_kernelbase && (eprom_kernelbase != kernelbase))
1565                 cmn_err(CE_WARN, "kernelbase value, User specified 0x%lx, "
1566                     "System using 0x%lx",
1567                     (uintptr_t)eprom_kernelbase, (uintptr_t)kernelbase);
1568 #endif

1570 #ifdef  KERNELBASE_ABI_MIN
1571         if (kernelbase < (uintptr_t)KERNELBASE_ABI_MIN) {
1572                 cmn_err(CE_NOTE, "!kernelbase set to 0x%lx, system is not "
1573                     "i386 ABI compliant.", (uintptr_t)kernelbase);
1574         }
1575 #endif

1577 #ifndef __xpv
1578         if (plat_dr_support_memory()) {
1579                 mem_config_init();
1580         }
1581 #else   /* __xpv */
1582         /*
1583          * Some of the xen start information has to be relocated up
1584          * into the kernel's permanent address space.
1585          */
1586         PRM_POINT("calling xen_relocate_start_info()");
1587         xen_relocate_start_info();
1588         PRM_POINT("xen_relocate_start_info() done");

1590         /*
1591          * (Update the vcpu pointer in our cpu structure to point into
1592          * the relocated shared info.)
1593          */
1594         CPU->cpu_m.mcpu_vcpu_info =
1595             &HYPERVISOR_shared_info->vcpu_info[CPU->cpu_id];
1596 #endif  /* __xpv */

1598         PRM_POINT("startup_kmem() done");
1599 }
```

unchanged_portion_omitted_

```
2013 /*
2014  * Finish initializing the VM system, now that we are no longer
2015  * relying on the boot time memory allocators.
2016  */
2017 static void
2018 startup_vm(void)
2019 {
2020         struct segmap_crargs a;

2022         extern int use_brk_lpg, use_stk_lpg;

2024         PRM_POINT("startup_vm() starting...");

2026         /*
2027          * Initialize the hat layer.
2028          */
2029         hat_init();

2031         /*
2032          * Do final allocations of HAT data structures that need to
2033          * be allocated before quiescing the boot loader.
2034          */
2035         PRM_POINT("Calling hat_kern_alloc()...");
2036         hat_kern_alloc((caddr_t)segmap_start, segmapsize, ekernelheap);
2037         PRM_POINT("hat_kern_alloc() done");
```



```
2039 #ifndef __xpv
2040         /*
2041          * Setup Page Attribute Table
2042          */
2043         pat_sync();
2044 #endif

2046         /*
2047          * The next two loops are done in distinct steps in order
2048          * to be sure that any page that is doubly mapped (both above
2049          * KERNEL_TEXT and below kernelbase) is dealt with correctly.
2050          * Note this may never happen, but it might someday.
2051          */
2052         bootpages = NULL;
2053         PRM_POINT("Protecting boot pages");

2055         /*
2056          * Protect any pages mapped above KERNEL_TEXT that somehow have
2057          * page_t's. This can only happen if something weird allocated
2058          * in this range (like kadb/kmdb).
2059          */
2060         protect_boot_range(KERNEL_TEXT, (uintptr_t)-1, 0);

2062         /*
2063          * Before we can take over memory allocation/mapping from the boot
2064          * loader we must remove from our free page lists any boot allocated
2065          * pages that stay mapped until release_bootstrap().
2066          */
2067         protect_boot_range(0, kernelbase, 1);


2070         /*
2071          * Switch to running on regular HAT (not boot_mmu)
2072          */
2073         PRM_POINT("Calling hat_kern_setup()...");
2074         hat_kern_setup();

2076         /*
2077          * It is no longer safe to call BOP_ALLOC(), so make sure we don't.
2078          */
2079         bop_no_more_mem();

2081         PRM_POINT("hat_kern_setup() done");

2083         hat_cpu_online(CPU);

2085         /*
2086          * Initialize VM system
2087          */
2088         PRM_POINT("Calling kvm_init()...");
2089         kvm_init();
2090         PRM_POINT("kvm_init() done");

2092         /*
2093          * Tell kmdb that the VM system is now working
2094          */
2095         if (boothowto & RB_DEBUG)
2096                 kdi_dvec_vmready();

2098 #if defined(__xpv)
2099         /*
2100          * Populate the I/O pool on domain 0
2101          */
2102         if (DOMAIN_IS_INITDOMAIN(xen_info)) {
2103                 extern long populate_io_pool(void);
2104                 long init_io_pool_cnt;
```

```
2106                 PRM_POINT("Populating reserve I/O page pool");
2107                 init_io_pool_cnt = populate_io_pool();
2108                 PRM_DEBUG(init_io_pool_cnt);
2109         }
2110 #endif
2111         /*
2112          * Mangle the brand string etc.
2113          */
2114         cpuid_pass3(CPU);

2116 #if defined(__amd64)

2118         /*
2119          * Create the device arena for toxic (to dtrace/kmdb) mappings.
2120          */
2121         device_arena = vmem_create("device", (void *)toxic_addr,
2122             toxic_size, MMU_PAGESIZE, NULL, NULL, NULL, 0, VM_SLEEP);

2124 #else   /* __i386 */

2126         /*
2127          * allocate the bit map that tracks toxic pages
2128          */
2129         toxic_bit_map_len = btop((ulong_t)(valloc_base - kernelbase));
2130         PRM_DEBUG(toxic_bit_map_len);
2131         toxic_bit_map =
2132             kmem_zalloc(BT_SIZEOFMAP(toxic_bit_map_len), KM_NOSLEEP);
2133         ASSERT(toxic_bit_map != NULL);
2134         PRM_DEBUG(toxic_bit_map);

2136 #endif  /* __i386 */


2139         /*
2140          * Now that we've got more VA, as well as the ability to allocate from
2141          * it, tell the debugger.
2142          */
2143         if (boothowto & RB_DEBUG)
2144                 kdi_dvec_memavail();

2124         /*
2125          * The following code installs a special page fault handler (#pf)
2126          * to work around a pentium bug.
2127          */
2128 #if !defined(__amd64) && !defined(__xpv)
2129         if (x86_type == X86_TYPE_P5) {
2130                 desctbr_t idtr;
2131                 gate_desc_t *newidt;

2133                 if ((newidt = kmem_zalloc(MMU_PAGESIZE, KM_NOSLEEP)) == NULL)
2134                         panic("failed to install pentium_pftrap");

2136                 bcopy(idt0, newidt, NIDT * sizeof (*idt0));
2137                 set_gatesegd(&newidt[T_PGFLT], &pentium_pftrap,
2138                     KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);

2140                 (void) as_setprot(&kas, (caddr_t)newidt, MMU_PAGESIZE,
2141                     PROT_READ | PROT_EXEC);

2143                 CPU->cpu_idt = newidt;
2144                 idtr.dtr_base = (uintptr_t)CPU->cpu_idt;
2145                 idtr.dtr_limit = (NIDT * sizeof (*idt0)) - 1;
2146                 wr_idtr(&idtr);
2147         }
2148 #endif  /* !__amd64 */
```



```
2146 #if !defined(__xpv)
2147         /*
2148          * Map page pfn=0 for drivers, such as kd, that need to pick up
2149          * parameters left there by controllers/BIOS.
2150          */
2151         PRM_POINT("setup up p0_va");
2152         p0_va = i86devmap(0, 1, PROT_READ);
2153         PRM_DEBUG(p0_va);
2154 #endif

2156         cmn_err(CE_CONT, "?mem = %luK (0x%lx)\n",
2157             physinstalled << (MMU_PAGESHIFT - 10), ptob(physinstalled));

2159         /*
2160          * disable automatic large pages for small memory systems or
2161          * when the disable flag is set.
2162          *
2163          * Do not yet consider page sizes larger than 2m/4m.
2164          */
2165         if (!auto_lpg_disable && mmu.max_page_level > 0) {
2166                 max_uheap_lpsize = LEVEL_SIZE(1);
2167                 max_ustack_lpsize = LEVEL_SIZE(1);
2168                 max_privmap_lpsize = LEVEL_SIZE(1);
2169                 max_uidata_lpsize = LEVEL_SIZE(1);
2170                 max_utext_lpsize = LEVEL_SIZE(1);
2171                 max_shm_lpsize = LEVEL_SIZE(1);
2172         }
2173         if (physmem < privm_lpg_min_physmem || mmu.max_page_level == 0 ||
2174             auto_lpg_disable) {
2175                 use_brk_lpg = 0;
2176                 use_stk_lpg = 0;
2177         }
2178         mcntl0_lpsize = LEVEL_SIZE(mmu.umax_page_level);

2180         PRM_POINT("Calling hat_init_finish()...");
2181         hat_init_finish();
2182         PRM_POINT("hat_init_finish() done");

2184         /*
2185          * Initialize the segkp segment type.
2186          */
2187         rw_enter(&kas.a_lock, RW_WRITER);
2188         PRM_POINT("Attaching segkp");
2189         if (segkp_fromheap) {
2190                 segkp->s_as = &kas;
2191         } else if (seg_attach(&kas, (caddr_t)segkp_base, mmu_ptob(segkpsize),
2192             segkp) < 0) {
2193                 panic("startup: cannot attach segkp");
2194                 /*NOTREACHED*/
2195         }
2196         PRM_POINT("Doing segkp_create()");
2197         if (segkp_create(segkp) != 0) {
2198                 panic("startup: segkp_create failed");
2199                 /*NOTREACHED*/
2200         }
2201         PRM_DEBUG(segkp);
2202         rw_exit(&kas.a_lock);

2204         /*
2205          * kpm segment
2206          */
2207         segmap_kpm = 0;
2208         if (kpm_desired)
2212         if (kpm_desired) {
2209                 kpm_init();
```

```
2214                    kpm_enable = 1;
2215            }

2211            /*
2212             * Now create segmap segment.
2213             */
2214            rw_enter(&kas.a_lock, RW_WRITER);
2215            if (seg_attach(&kas, (caddr_t)segmap_start, segmapsize, segmap) < 0) {
2216                    panic("cannot attach segmap");
2217                    /*NOTREACHED*/
2218            }
2219            PRM_DEBUG(segmap);

2221            a.prot = PROT_READ | PROT_WRITE;
2222            a.shmsize = 0;
2223            a.nfreelist = segmapfreelists;

2225            if (segmap_create(segmap, (caddr_t)&a) != 0)
2226                    panic("segmap_create segmap");
2227            rw_exit(&kas.a_lock);

2229            setup_vaddr_for_ppcopy(CPU);

2231            segdev_init();
2232 #if defined(__xpv)
2233            if (DOMAIN_IS_INITDOMAIN(xen_info))
2234 #endif
2235                    pmem_init();

2237            PRM_POINT("startup_vm() done");
2238 }
_____unchanged_portion_omitted_

2250 static void
2251 startup_end(void)
2252 {
2253            int i;
2254            extern void setx86isalist(void);
2255            extern void cpu_event_init(void);

2257            PRM_POINT("startup_end() starting...");

2259            /*
2260             * Perform tasks that get done after most of the VM
2261             * initialization has been done but before the clock
2262             * and other devices get started.
2263             */
2264            kern_setup1();

2266            /*
2267             * Perform CPC initialization for this CPU.
2268             */
2269            kcpc_hw_init(CPU);

2271            /*
2272             * Initialize cpu event framework.
2273             */
2274            cpu_event_init();

2276 #if defined(OPTERON_WORKAROUND_6323525)
2277            if (opteron_workaround_6323525)
2278                    patch_workaround_6323525();
2279 #endif
2280            /*
2281             * If needed, load TOD module now so that ddi_get_time(9F) etc. work
2282             * (For now, "needed" is defined as set tod_module_name in /etc/system)
```



```
2283             */
2284            if (tod_module_name != NULL) {
2285                    PRM_POINT("load_tod_module()");
2286                    load_tod_module(tod_module_name);
2287            }

2289 #if defined(__xpv)
2290            /*
2291             * Forceload interposing TOD module for the hypervisor.
2292             */
2293            PRM_POINT("load_tod_module()");
2294            load_tod_module("xpvtod");
2295 #endif

2297            /*
2298             * Configure the system.
2299             */
2300            PRM_POINT("Calling configure()...");
2301            configure();            /* set up devices */
2302            PRM_POINT("configure() done");

2304            /*
2305             * We can now setup for XSAVE because fpu_probe is done in configure().
2306             */
2307            if (fp_save_mech == FP_XSAVE) {
2308                    xsave_setup_msr(CPU);
2309            }

2311            /*
2312             * Set the isa_list string to the defined instruction sets we
2313             * support.
2314             */
2315            setx86isalist();
2316            cpu_intr_alloc(CPU, NINTR_THREADS);
2317            psm_install();

2319            /*
2320             * We're done with bootops.  We don't unmap the bootstrap yet because
2321             * we're still using bootsvcs.
2322             */
2323            PRM_POINT("NULLing out bootops");
2324            *bootopsp = (struct bootops *)NULL;
2325            bootops = (struct bootops *)NULL;

2327 #if defined(__xpv)
2328            ec_init_debug_irq();
2329            xs_domu_init();
2330 #endif

2332 #if !defined(__xpv)
2338 #if defined(__amd64) && !defined(__xpv)
2333            /*
2334             * Intel IOMMU has been setup/initialized in ddi_impl.c
2335             * Start it up now.
2336             */
2337            immu_startup();

2339            /*
2340             * Now that we're no longer going to drop into real mode for a BIOS call
2341             * via bootops, we can enable PCID (which requires CR0.PG).
2342             */
2343            enable_pcid();
2344 #endif

2346            PRM_POINT("Enabling interrupts");
2347            (*picinitf)();
```

```
2348         sti();
2349 #if defined(__xpv)
2350         ASSERT(CPU->cpu_m.mcpu_vcpu_info->evtchn_upcall_mask == 0);
2351         xen_late_startup();
2352 #endif

2354         (void) add_avsoftintr((void *)&softlevel1_hdl, 1, softlevel1,
2355             "softlevel1", NULL, NULL); /* XXX to be moved later */

2357         /*
2358          * Register software interrupt handlers for ddi_periodic_add(9F).
2359          * Software interrupts up to the level 10 are supported.
2360          */
2361         for (i = DDI_IPL_1; i <= DDI_IPL_10; i++) {
2362                 (void) add_avsoftintr((void *)&softlevel_hdl[i-1], i,
2363                     (avfunc)ddi_periodic_softintr, "ddi_periodic",
2364                     (caddr_t)(uintptr_t)i, NULL);
2365         }

2367 #if !defined(__xpv)
2368         if (modload("drv", "amd_iommu") < 0) {
2369                 PRM_POINT("No AMD IOMMU present\n");
2370         } else if (ddi_hold_installed_driver(ddi_name_to_major(
2371             "amd_iommu")) == NULL) {
2372                 prom_printf("ERROR: failed to attach AMD IOMMU\n");
2373         }
2374 #endif
2375         post_startup_cpu_fixups();

2377         PRM_POINT("startup_end() done");
2378 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   64156 Fri Apr  6 17:25:03 2018
new/usr/src/uts/i86pc/os/trap.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*      Copyright (c) 1990, 1991 UNIX System Laboratories, Inc. */
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989, 1990 AT&T   */
/*              All Rights Reserved                             */
/*                                                              */
/*      Copyright (c) 1987, 1988 Microsoft Corporation          */
/*              All Rights Reserved                             */
/*                                                              */

/*
 * Copyright 2018 Joyent, Inc.
 * Copyright 2017 Joyent, Inc.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/param.h>
#include <sys/signal.h>
#include <sys/systm.h>
#include <sys/user.h>
#include <sys/proc.h>
#include <sys/disp.h>
#include <sys/class.h>
#include <sys/core.h>
#include <sys/syscall.h>
#include <sys/cpuvar.h>
#include <sys/vm.h>
#include <sys/sysinfo.h>
#include <sys/fault.h>
#include <sys/stack.h>
#include <sys/psw.h>
#include <sys/regset.h>
#include <sys/fp.h>
#include <sys/trap.h>
#include <sys/kmem.h>
```



```
#include <sys/vtrace.h>
#include <sys/cmn_err.h>
#include <sys/prsystm.h>
#include <sys/mutex_impl.h>
#include <sys/machsystm.h>
#include <sys/archsystm.h>
#include <sys/sdt.h>
#include <sys/avintr.h>
#include <sys/kobj.h>

#include <vm/hat.h>

#include <vm/seg_kmem.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/hat_pte.h>
#include <vm/hat_i86.h>

#include <sys/procfs.h>

#include <sys/reboot.h>
#include <sys/debug.h>
#include <sys/debugreg.h>
#include <sys/modctl.h>
#include <sys/aio_impl.h>
#include <sys/tnf.h>
#include <sys/tnf_probe.h>
#include <sys/cred.h>
#include <sys/mman.h>
#include <sys/x86_archext.h>
#include <sys/copyops.h>
#include <c2/audit.h>
#include <sys/ftrace.h>
#include <sys/panic.h>
#include <sys/traptrace.h>
#include <sys/ontrap.h>
#include <sys/cpc_impl.h>
#include <sys/bootconf.h>
#include <sys/bootinfo.h>
#include <sys/promif.h>
#include <sys/mach_mmu.h>
#if defined(__xpv)
#include <sys/hypervisor.h>
#endif
#include <sys/contract/process_impl.h>

#define USER    0x10000         /* user-mode flag added to trap type */

static const char *trap_type_mnemonic[] = {
        "de",   "db",   "2",    "bp",
        "of",   "br",   "ud",   "nm",
        "df",   "9",    "ts",   "np",
        "ss",   "gp",   "pf",   "15",
        "mf",   "ac",   "mc",   "xf"
};
```

_____*unchanged_portion_omitted_*

```
#endif /* OPTERON_ERRATUM_91 */

/*
 * Called from the trap handler when a processor trap occurs.
 *
 * Note: All user-level traps that might call stop() must exit
 * trap() by 'goto out' or by falling through.
 * Note Also: trap() is usually called with interrupts enabled, (PS_IE == 1)
 * however, there are paths that arrive here with PS_IE == 0 so special care
```

```
 462  * must be taken in those cases.
 463  */
 464 void
 465 trap(struct regs *rp, caddr_t addr, processorid_t cpuid)
 466 {
 467         kthread_t *ct = curthread;
 468         enum seg_rw rw;
 469         unsigned type;
 470         proc_t *p = ttoproc(ct);
 471         klwp_t *lwp = ttolwp(ct);
 472         uintptr_t lofault;
 473         label_t *onfault;
 474         faultcode_t pagefault(), res, errcode;
 475         enum fault_type fault_type;
 476         k_siginfo_t siginfo;
 477         uint_t fault = 0;
 478         int mstate;
 479         int sicode = 0;
 480         int watchcode;
 481         int watchpage;
 482         caddr_t vaddr;
 483         int singlestep_twiddle;
 483         size_t sz;
 484         int ta;
 485 #ifdef __amd64
 486         uchar_t instr;
 487 #endif

 489         ASSERT_STACK_ALIGNED();

 491         type = rp->r_trapno;
 492         CPU_STATS_ADDQ(CPU, sys, trap, 1);
 493         ASSERT(ct->t_schedflag & TS_DONT_SWAP);

 495         if (type == T_PGFLT) {

 497                 errcode = rp->r_err;
 498                 if (errcode & PF_ERR_WRITE)
 499                         rw = S_WRITE;
 500                 else if ((caddr_t)rp->r_pc == addr ||
 501                     (mmu.pt_nx != 0 && (errcode & PF_ERR_EXEC)))
 502                         rw = S_EXEC;
 503                 else
 504                         rw = S_READ;

 506 #if defined(__i386)
 507                 /*
 508                  * Pentium Pro work-around
 509                  */
 510                 if ((errcode & PF_ERR_PROT) && pentiumpro_bug4046376) {
 511                         uint_t  attr;
 512                         uint_t  priv_violation;
 513                         uint_t  access_violation;

 515                         if (hat_getattr(addr < (caddr_t)kernelbase ?
 516                             curproc->p_as->a_hat : kas.a_hat, addr, &attr)
 517                             == -1) {
 518                                 errcode &= ~PF_ERR_PROT;
 519                         } else {
 520                                 priv_violation = (errcode & PF_ERR_USER) &&
 521                                     !(attr & PROT_USER);
 522                                 access_violation = (errcode & PF_ERR_WRITE) &&
 523                                     !(attr & PROT_WRITE);
 524                                 if (!priv_violation && !access_violation)
 525                                         goto cleanup;
 526                         }
```



```
 527                 }
 528 #endif /* __i386 */

 530         } else if (type == T_SGLSTP && lwp != NULL)
 531                 lwp->lwp_pcb.pcb_drstat = (uintptr_t)addr;

 533         if (tdebug)
 534                 showregs(type, rp, addr);

 536         if (USERMODE(rp->r_cs)) {
 537                 /*
 538                  * Set up the current cred to use during this trap. u_cred
 539                  * no longer exists.  t_cred is used instead.
 540                  * The current process credential applies to the thread for
 541                  * the entire trap.  If trapping from the kernel, this
 542                  * should already be set up.
 543                  */
 544                 if (ct->t_cred != p->p_cred) {
 545                         cred_t *oldcred = ct->t_cred;
 546                         /*
 547                          * DTrace accesses t_cred in probe context.  t_cred
 548                          * must always be either NULL, or point to a valid,
 549                          * allocated cred structure.
 550                          */
 551                         ct->t_cred = crgetcred();
 552                         crfree(oldcred);
 553                 }
 554                 ASSERT(lwp != NULL);
 555                 type |= USER;
 556                 ASSERT(lwptoregs(lwp) == rp);
 557                 lwp->lwp_state = LWP_SYS;

 559                 switch (type) {
 560                 case T_PGFLT + USER:
 561                         if ((caddr_t)rp->r_pc == addr)
 562                                 mstate = LMS_TFAULT;
 563                         else
 564                                 mstate = LMS_DFAULT;
 565                         break;
 566                 default:
 567                         mstate = LMS_TRAP;
 568                         break;
 569                 }
 570                 /* Kernel probe */
 571                 TNF_PROBE_1(thread_state, "thread", /* CSTYLED */,
 572                     tnf_microstate, state, mstate);
 573                 mstate = new_mstate(ct, mstate);

 575                 bzero(&siginfo, sizeof (siginfo));
 576         }

 578         switch (type) {
 579         case T_PGFLT + USER:
 580         case T_SGLSTP:
 581         case T_SGLSTP + USER:
 582         case T_BPTFLT + USER:
 583                 break;

 585         default:
 586                 FTRACE_2("trap(): type=0x%lx, regs=0x%lx",
 587                     (ulong_t)type, (ulong_t)rp);
 588                 break;
 589         }

 591         switch (type) {
 592         case T_SIMDFPE:
```

```
593                 /* Make sure we enable interrupts before die()ing */
594                 sti();  /* The SIMD exception comes in via cmninttrap */
595                 /*FALLTHROUGH*/
596         default:
597                 if (type & USER) {
598                         if (tudebug)
599                                 showregs(type, rp, (caddr_t)0);
600                         printf("trap: Unknown trap type %d in user mode\n",
601                             type & ~USER);
602                         siginfo.si_signo = SIGILL;
603                         siginfo.si_code  = ILL_ILLTRP;
604                         siginfo.si_addr  = (caddr_t)rp->r_pc;
605                         siginfo.si_trapno = type & ~USER;
606                         fault = FLTILL;
607                         break;
608                 } else {
609                         (void) die(type, rp, addr, cpuid);
610                         /*NOTREACHED*/
611                 }
612
613         case T_PGFLT:           /* system page fault */
614                 /*
615                  * If we're under on_trap() protection (see <sys/ontrap.h>),
616                  * set ot_trap and bounce back to the on_trap() call site
617                  * via the installed trampoline.
618                  */
619                 if ((ct->t_ontrap != NULL) &&
620                     (ct->t_ontrap->ot_prot & OT_DATA_ACCESS)) {
621                         ct->t_ontrap->ot_trap |= OT_DATA_ACCESS;
622                         rp->r_pc = ct->t_ontrap->ot_trampoline;
623                         goto cleanup;
624                 }
625
626                 /*
627                  * If we have an Instruction fault in kernel mode, then that
628                  * means we've tried to execute a user page (SMEP) or both of
629                  * PAE and NXE are enabled. In either case, given that it's a
630                  * kernel fault, we should panic immediately and not try to make
631                  * any more forward progress. This indicates a bug in the
632                  * kernel, which if execution continued, could be exploited to
633                  * wreak havoc on the system.
634                  */
635                 if (errcode & PF_ERR_EXEC) {
636                         (void) die(type, rp, addr, cpuid);
637                 }
638
639                 /*
640                  * We need to check if SMAP is in play. If SMAP is in play, then
641                  * any access to a user page will show up as a protection
642                  * violation. To see if SMAP is enabled we first check if it's a
643                  * user address and whether we have the feature flag set. If we
644                  * do and the interrupted registers do not allow for user
645                  * accesses (PS_ACHK is not enabled), then we need to die
646                  * immediately.
647                  */
648                 if (addr < (caddr_t)kernelbase &&
649                     is_x86_feature(x86_featureset, X86FSET_SMAP) == B_TRUE &&
650                     (rp->r_ps & PS_ACHK) == 0) {
651                         (void) die(type, rp, addr, cpuid);
652                 }
653
654                 /*
655                  * See if we can handle as pagefault. Save lofault and onfault
656                  * across this. Here we assume that an address less than
657                  * KERNELBASE is a user fault.  We can do this as copy.s
658                  * routines verify that the starting address is less than
```



```
659                  * KERNELBASE before starting and because we know that we
660                  * always have KERNELBASE mapped as invalid to serve as a
661                  * "barrier".
662                  */
663                 lofault = ct->t_lofault;
664                 onfault = ct->t_onfault;
665                 ct->t_lofault = 0;
666
667                 mstate = new_mstate(ct, LMS_KFAULT);
668
669                 if (addr < (caddr_t)kernelbase) {
670                         res = pagefault(addr,
671                             (errcode & PF_ERR_PROT)? F_PROT: F_INVAL, rw, 0);
672                         if (res == FC_NOMAP &&
673                             addr < p->p_usrstack &&
674                             grow(addr))
675                                 res = 0;
676                 } else {
677                         res = pagefault(addr,
678                             (errcode & PF_ERR_PROT)? F_PROT: F_INVAL, rw, 1);
679                 }
680                 (void) new_mstate(ct, mstate);
681
682                 /*
683                  * Restore lofault and onfault. If we resolved the fault, exit.
684                  * If we didn't and lofault wasn't set, die.
685                  */
686                 ct->t_lofault = lofault;
687                 ct->t_onfault = onfault;
688                 if (res == 0)
689                         goto cleanup;
690
691 #if defined(OPTERON_ERRATUM_93) && defined(_LP64)
692                 if (lofault == 0 && opteron_erratum_93) {
693                         /*
694                          * Workaround for Opteron Erratum 93. On return from
695                          * a System Managment Interrupt at a HLT instruction
696                          * the %rip might be truncated to a 32 bit value.
697                          * BIOS is supposed to fix this, but some don't.
698                          * If this occurs we simply restore the high order bits.
699                          * The HLT instruction is 1 byte of 0xf4.
700                          */
701                         uintptr_t       rip = rp->r_pc;
702
703                         if ((rip & 0xffffffffUL) == rip) {
704                                 rip |= 0xffffffffUL << 32;
705                                 if (hat_getpfnum(kas.a_hat, (caddr_t)rip) !=
706                                     PFN_INVALID &&
707                                     (*(uchar_t *)rip == 0xf4 ||
708                                     *(uchar_t *)(rip - 1) == 0xf4)) {
709                                         rp->r_pc = rip;
710                                         goto cleanup;
711                                 }
712                         }
713                 }
714 #endif /* OPTERON_ERRATUM_93 && _LP64 */
715
716 #ifdef OPTERON_ERRATUM_91
717                 if (lofault == 0 && opteron_erratum_91) {
718                         /*
719                          * Workaround for Opteron Erratum 91. Prefetches may
720                          * generate a page fault (they're not supposed to do
721                          * that!). If this occurs we simply return back to the
722                          * instruction.
723                          */
724                         caddr_t         pc = (caddr_t)rp->r_pc;
```

```
726                         /*
727                          * If the faulting PC is not mapped, this is a
728                          * legitimate kernel page fault that must result in a
729                          * panic. If the faulting PC is mapped, it could contain
730                          * a prefetch instruction. Check for that here.
731                          */
732                         if (hat_getpfnum(kas.a_hat, pc) != PFN_INVALID) {
733                                 if (cmp_to_prefetch((uchar_t *)pc)) {
734 #ifdef DEBUG
735                                         cmn_err(CE_WARN, "Opteron erratum 91 "
736                                             "occurred: kernel prefetch"
737                                             " at %p generated a page fault!",
738                                             (void *)rp->r_pc);
739 #endif /* DEBUG */
740                                         goto cleanup;
741                                 }
742                         }
743                         (void) die(type, rp, addr, cpuid);
744                 }
745 #endif /* OPTERON_ERRATUM_91 */

747                 if (lofault == 0)
748                         (void) die(type, rp, addr, cpuid);

750                 /*
751                  * Cannot resolve fault.  Return to lofault.
752                  */
753                 if (lodebug) {
754                         showregs(type, rp, addr);
755                         traceregs(rp);
756                 }
757                 if (FC_CODE(res) == FC_OBJERR)
758                         res = FC_ERRNO(res);
759                 else
760                         res = EFAULT;
761                 rp->r_r0 = res;
762                 rp->r_pc = ct->t_lofault;
763                 goto cleanup;

765         case T_PGFLT + USER:    /* user page fault */
766                 if (faultdebug) {
767                         char *fault_str;

769                         switch (rw) {
770                         case S_READ:
771                                 fault_str = "read";
772                                 break;
773                         case S_WRITE:
774                                 fault_str = "write";
775                                 break;
776                         case S_EXEC:
777                                 fault_str = "exec";
778                                 break;
779                         default:
780                                 fault_str = "";
781                                 break;
782                         }
783                         printf("user %s fault:  addr=0x%lx errcode=0x%x\n",
784                             fault_str, (uintptr_t)addr, errcode);
785                 }

787 #if defined(OPTERON_ERRATUM_100) && defined(_LP64)
788                 /*
789                  * Workaround for AMD erratum 100
790                  *
```



```
791                  * A 32-bit process may receive a page fault on a non
792                  * 32-bit address by mistake. The range of the faulting
793                  * address will be
794                  *
795                  *      0xffffffff80000000 .. 0xffffffffffffffff or
796                  *      0x0000000100000000 .. 0x000000017fffffff
797                  *
798                  * The fault is always due to an instruction fetch, however
799                  * the value of r_pc should be correct (in 32 bit range),
800                  * so we ignore the page fault on the bogus address.
801                  */
802                 if (p->p_model == DATAMODEL_ILP32 &&
803                     (0xffffffff80000000 <= (uintptr_t)addr ||
804                     (0x100000000 <= (uintptr_t)addr &&
805                     (uintptr_t)addr <= 0x17fffffff))) {
806                         if (!opteron_erratum_100)
807                                 panic("unexpected erratum #100");
808                         if (rp->r_pc <= 0xffffffff)
809                                 goto out;
810                 }
811 #endif /* OPTERON_ERRATUM_100 && _LP64 */

813                 ASSERT(!(curthread->t_flag & T_WATCHPT));
814                 watchpage = (pr_watch_active(p) && pr_is_watchpage(addr, rw));
815 #ifdef __i386
816                 /*
817                  * In 32-bit mode, the lcall (system call) instruction fetches
818                  * one word from the stack, at the stack pointer, because of the
819                  * way the call gate is constructed.  This is a bogus
820                  * read and should not be counted as a read watchpoint.
821                  * We work around the problem here by testing to see if
822                  * this situation applies and, if so, simply jumping to
823                  * the code in locore.s that fields the system call trap.
824                  * The registers on the stack are already set up properly
825                  * due to the match between the call gate sequence and the
826                  * trap gate sequence.  We just have to adjust the pc.
827                  */
828                 if (watchpage && addr == (caddr_t)rp->r_sp &&
829                     rw == S_READ && instr_is_lcall_syscall((caddr_t)rp->r_pc)) {
830                         extern void watch_syscall(void);

832                         rp->r_pc += LCALLSIZE;
833                         watch_syscall();        /* never returns */
834                         /* NOTREACHED */
835                 }
836 #endif /* __i386 */
837                 vaddr = addr;
838                 if (!watchpage || (sz = instr_size(rp, &vaddr, rw)) <= 0)
839                         fault_type = (errcode & PF_ERR_PROT)? F_PROT: F_INVAL;
840                 else if ((watchcode = pr_is_watchpoint(&vaddr, &ta,
841                     sz, NULL, rw)) != 0) {
842                         if (ta) {
843                                 do_watch_step(vaddr, sz, rw,
844                                     watchcode, rp->r_pc);
845                                 fault_type = F_INVAL;
846                         } else {
847                                 bzero(&siginfo, sizeof (siginfo));
848                                 siginfo.si_signo = SIGTRAP;
849                                 siginfo.si_code = watchcode;
850                                 siginfo.si_addr = vaddr;
851                                 siginfo.si_trapafter = 0;
852                                 siginfo.si_pc = (caddr_t)rp->r_pc;
853                                 fault = FLTWATCH;
854                                 break;
855                         }
856                 } else {
```

```
857                         /* XXX pr_watch_emul() never succeeds (for now) */
858                         if (rw != S_EXEC && pr_watch_emul(rp, vaddr, rw))
859                                 goto out;
860                         do_watch_step(vaddr, sz, rw, 0, 0);
861                         fault_type = F_INVAL;
862                 }

864                 res = pagefault(addr, fault_type, rw, 0);

866                 /*
867                  * If pagefault() succeeded, ok.
868                  * Otherwise attempt to grow the stack.
869                  */
870                 if (res == 0 ||
871                     (res == FC_NOMAP &&
872                     addr < p->p_usrstack &&
873                     grow(addr))) {
874                         lwp->lwp_lastfault = FLTPAGE;
875                         lwp->lwp_lastfaddr = addr;
876                         if (prismember(&p->p_fltmask, FLTPAGE)) {
877                                 bzero(&siginfo, sizeof (siginfo));
878                                 siginfo.si_addr = addr;
879                                 (void) stop_on_fault(FLTPAGE, &siginfo);
880                         }
881                         goto out;
882                 } else if (res == FC_PROT && addr < p->p_usrstack &&
883                     (mmu.pt_nx != 0 && (errcode & PF_ERR_EXEC))) {
884                         report_stack_exec(p, addr);
885                 }

887 #ifdef OPTERON_ERRATUM_91
888                 /*
889                  * Workaround for Opteron Erratum 91. Prefetches may generate a
890                  * page fault (they're not supposed to do that!). If this
891                  * occurs we simply return back to the instruction.
892                  *
893                  * We rely on copyin to properly fault in the page with r_pc.
894                  */
895                 if (opteron_erratum_91 &&
896                     addr != (caddr_t)rp->r_pc &&
897                     instr_is_prefetch((caddr_t)rp->r_pc)) {
898 #ifdef DEBUG
899                         cmn_err(CE_WARN, "Opteron erratum 91 occurred: "
900                             "prefetch at %p in pid %d generated a trap!",
901                             (void *)rp->r_pc, p->p_pid);
902 #endif /* DEBUG */
903                         goto out;
904                 }
905 #endif /* OPTERON_ERRATUM_91 */

907                 if (tudebug)
908                         showregs(type, rp, addr);
909                 /*
910                  * In the case where both pagefault and grow fail,
911                  * set the code to the value provided by pagefault.
912                  * We map all errors returned from pagefault() to SIGSEGV.
913                  */
914                 bzero(&siginfo, sizeof (siginfo));
915                 siginfo.si_addr = addr;
916                 switch (FC_CODE(res)) {
917                 case FC_HWERR:
918                 case FC_NOSUPPORT:
919                         siginfo.si_signo = SIGBUS;
920                         siginfo.si_code = BUS_ADRERR;
921                         fault = FLTACCESS;
922                         break;
```



```
923                 case FC_ALIGN:
924                         siginfo.si_signo = SIGBUS;
925                         siginfo.si_code = BUS_ADRALN;
926                         fault = FLTACCESS;
927                         break;
928                 case FC_OBJERR:
929                         if ((siginfo.si_errno = FC_ERRNO(res)) != EINTR) {
930                                 siginfo.si_signo = SIGBUS;
931                                 siginfo.si_code = BUS_OBJERR;
932                                 fault = FLTACCESS;
933                         }
934                         break;
935                 default:        /* FC_NOMAP or FC_PROT */
936                         siginfo.si_signo = SIGSEGV;
937                         siginfo.si_code =
938                             (res == FC_NOMAP)? SEGV_MAPERR : SEGV_ACCERR;
939                         fault = FLTBOUNDS;
940                         break;
941                 }
942                 break;

944         case T_ILLINST + USER:  /* invalid opcode fault */
945                 /*
946                  * If the syscall instruction is disabled due to LDT usage, a
947                  * user program that attempts to execute it will trigger a #ud
948                  * trap. Check for that case here. If this occurs on a CPU which
949                  * doesn't even support syscall, the result of all of this will
950                  * be to emulate that particular instruction.
951                  */
952                 if (p->p_ldt != NULL &&
953                     ldt_rewrite_syscall(rp, p, X86FSET_ASYSC))
954                         goto out;

956 #ifdef __amd64
957                 /*
958                  * Emulate the LAHF and SAHF instructions if needed.
959                  * See the instr_is_lsahf function for details.
960                  */
961                 if (p->p_model == DATAMODEL_LP64 &&
962                     instr_is_lsahf((caddr_t)rp->r_pc, &instr)) {
963                         emulate_lsahf(rp, instr);
964                         goto out;
965                 }
966 #endif

968                 /*FALLTHROUGH*/

970                 if (tudebug)
971                         showregs(type, rp, (caddr_t)0);
972                 siginfo.si_signo = SIGILL;
973                 siginfo.si_code  = ILL_ILLOPC;
974                 siginfo.si_addr  = (caddr_t)rp->r_pc;
975                 fault = FLTILL;
976                 break;

978         case T_ZERODIV + USER:          /* integer divide by zero */
979                 if (tudebug && tudebugfpe)
980                         showregs(type, rp, (caddr_t)0);
981                 siginfo.si_signo = SIGFPE;
982                 siginfo.si_code  = FPE_INTDIV;
983                 siginfo.si_addr  = (caddr_t)rp->r_pc;
984                 fault = FLTIZDIV;
985                 break;

987         case T_OVFLW + USER:    /* integer overflow */
988                 if (tudebug && tudebugfpe)
```

```
 989                     showregs(type, rp, (caddr_t)0);
 990             siginfo.si_signo = SIGFPE;
 991             siginfo.si_code  = FPE_INTOVF;
 992             siginfo.si_addr  = (caddr_t)rp->r_pc;
 993             fault = FLTIOVF;
 994             break;

 996     case T_NOEXTFLT + USER: /* math coprocessor not available */
 997             if (tudebug && tudebugfpe)
 998                     showregs(type, rp, addr);
 999             if (fpnoextflt(rp)) {
1000                     siginfo.si_signo = SIGILL;
1001                     siginfo.si_code  = ILL_ILLOPC;
1002                     siginfo.si_addr  = (caddr_t)rp->r_pc;
1003                     fault = FLTILL;
1004             }
1005             break;

1007     case T_EXTOVRFLT:       /* extension overrun fault */
1008             /* check if we took a kernel trap on behalf of user */
1009             {
1010                     extern  void ndptrap_frstor(void);
1011                     if (rp->r_pc != (uintptr_t)ndptrap_frstor) {
1012                             sti(); /* T_EXTOVRFLT comes in via cmninttrap */
1013                             (void) die(type, rp, addr, cpuid);
1014                     }
1015                     type |= USER;
1016             }
1017             /*FALLTHROUGH*/
1018     case T_EXTOVRFLT + USER:        /* extension overrun fault */
1019             if (tudebug && tudebugfpe)
1020                     showregs(type, rp, addr);
1021             if (fpextovrflt(rp)) {
1022                     siginfo.si_signo = SIGSEGV;
1023                     siginfo.si_code  = SEGV_MAPERR;
1024                     siginfo.si_addr  = (caddr_t)rp->r_pc;
1025                     fault = FLTBOUNDS;
1026             }
1027             break;

1029     case T_EXTERRFLT:       /* x87 floating point exception pending */
1030             /* check if we took a kernel trap on behalf of user */
1031             {
1032                     extern  void ndptrap_frstor(void);
1033                     if (rp->r_pc != (uintptr_t)ndptrap_frstor) {
1034                             sti(); /* T_EXTERRFLT comes in via cmninttrap */
1035                             (void) die(type, rp, addr, cpuid);
1036                     }
1037                     type |= USER;
1038             }
1039             /*FALLTHROUGH*/

1041     case T_EXTERRFLT + USER: /* x87 floating point exception pending */
1042             if (tudebug && tudebugfpe)
1043                     showregs(type, rp, addr);
1044             if (sicode = fpexterrflt(rp)) {
1045                     siginfo.si_signo = SIGFPE;
1046                     siginfo.si_code  = sicode;
1047                     siginfo.si_addr  = (caddr_t)rp->r_pc;
1048                     fault = FLTFPE;
1049             }
1050             break;

1052     case T_SIMDFPE + USER:          /* SSE and SSE2 exceptions */
1053             if (tudebug && tudebugsse)
1054                     showregs(type, rp, addr);
```



```
1055             if (!is_x86_feature(x86_featureset, X86FSET_SSE) &&
1056                 !is_x86_feature(x86_featureset, X86FSET_SSE2)) {
1057                     /*
1058                      * There are rumours that some user instructions
1059                      * on older CPUs can cause this trap to occur; in
1060                      * which case send a SIGILL instead of a SIGFPE.
1061                      */
1062                     siginfo.si_signo = SIGILL;
1063                     siginfo.si_code  = ILL_ILLTRP;
1064                     siginfo.si_addr  = (caddr_t)rp->r_pc;
1065                     siginfo.si_trapno = type & ~USER;
1066                     fault = FLTILL;
1067             } else if ((sicode = fpsimderrflt(rp)) != 0) {
1068                     siginfo.si_signo = SIGFPE;
1069                     siginfo.si_code = sicode;
1070                     siginfo.si_addr = (caddr_t)rp->r_pc;
1071                     fault = FLTFPE;
1072             }

1074             sti();  /* The SIMD exception comes in via cmninttrap */
1075             break;

1077     case T_BPTFLT:  /* breakpoint trap */
1078             /*
1079              * Kernel breakpoint traps should only happen when kmdb is
1080              * active, and even then, it'll have interposed on the IDT, so
1081              * control won't get here.  If it does, we've hit a breakpoint
1082              * without the debugger, which is very strange, and very
1083              * fatal.
1084              */
1085             if (tudebug && tudebugbpt)
1086                     showregs(type, rp, (caddr_t)0);

1088             (void) die(type, rp, addr, cpuid);
1089             break;

1091     case T_SGLSTP: /* single step/hw breakpoint exception */

1093 #if !defined(__xpv)
1094             /* Now evaluate how we got here */
1095             if (lwp != NULL && (lwp->lwp_pcb.pcb_drstat & DR_SINGLESTEP)) {
1094             /*
1095              * We'd never normally get here, as kmdb handles its own single
1096              * step traps.  There is one nasty exception though, as
1097              * described in more detail in sys_sysenter().  Note that
1098              * checking for all four locations covers both the KPTI and the
1099              * non-KPTI cases correctly: the former will never be found at
1100              * (brand_)sys_sysenter, and vice versa.
1097                      * i386 single-steps even through lcalls which
1098                      * change the privilege level. So we take a trap at
1099                      * the first instruction in privileged mode.
1100                      *
1101                      * Set a flag to indicate that upon completion of
1102                      * the system call, deal with the single-step trap.
1103                      *
1104                      * The same thing happens for sysenter, too.
1101              */
1102             if (lwp != NULL && (lwp->lwp_pcb.pcb_drstat & DR_SINGLESTEP)) {
1103                     if (rp->r_pc == (greg_t)brand_sys_sysenter ||
1104                         rp->r_pc == (greg_t)sys_sysenter ||
1105                         rp->r_pc == (greg_t)tr_brand_sys_sysenter ||
1106                         rp->r_pc == (greg_t)tr_sys_sysenter) {

1108                             rp->r_pc += 0x3; /* sizeof (swapgs) */

1106                     singlestep_twiddle = 0;
```

```
1107                        if (rp->r_pc == (uintptr_t)sys_sysenter ||
1108                            rp->r_pc == (uintptr_t)brand_sys_sysenter) {
1109                                singlestep_twiddle = 1;
1110 #if defined(__amd64)
1111                                /*
1112                                 * Since we are already on the kernel's
1113                                 * %gs, on 64-bit systems the sysenter case
1114                                 * needs to adjust the pc to avoid
1115                                 * executing the swapgs instruction at the
1116                                 * top of the handler.
1117                                 */
1118                                if (rp->r_pc == (uintptr_t)sys_sysenter)
1119                                        rp->r_pc = (uintptr_t)
1120                                            _sys_sysenter_post_swapgs;
1121                                else
1122                                        rp->r_pc = (uintptr_t)
1123                                            _brand_sys_sysenter_post_swapgs;
1124 #endif
1125                        }
1126 #if defined(__i386)
1127                        else if (rp->r_pc == (uintptr_t)sys_call ||
1128                            rp->r_pc == (uintptr_t)brand_sys_call) {
1129                                singlestep_twiddle = 1;
1130                        }
1131 #endif
1132                        else {
1133                                /* not on sysenter/syscall; uregs available */
1134                                if (tudebug && tudebugbpt)
1135                                        showregs(type, rp, (caddr_t)0);
1136                        }
1137                        if (singlestep_twiddle) {
1110                                rp->r_ps &= ~PS_T; /* turn off trace */
1111                                lwp->lwp_pcb.pcb_flags |= DEBUG_PENDING;
1112                                ct->t_post_sys = 1;
1113                                aston(curthread);
1114                                goto cleanup;
1115                        } else {
1116                                if (tudebug && tudebugbpt)
1117                                        showregs(type, rp, (caddr_t)0);
1118                        }
1119                }
1120 #endif /* !__xpv */

1145                /* XXX - needs review on debugger interface? */
1122                if (boothowto & RB_DEBUG)
1123                        debug_enter((char *)NULL);
1124                else
1125                        (void) die(type, rp, addr, cpuid);
1126                break;

1128        case T_NMIFLT:  /* NMI interrupt */
1129                printf("Unexpected NMI in system mode\n");
1130                goto cleanup;

1132        case T_NMIFLT + USER:   /* NMI interrupt */
1133                printf("Unexpected NMI in user mode\n");
1134                break;

1136        case T_GPFLT:   /* general protection violation */
1137                /*
1138                 * Any #GP that occurs during an on_trap .. no_trap bracket
1139                 * with OT_DATA_ACCESS or OT_SEGMENT_ACCESS protection,
1140                 * or in a on_fault .. no_fault bracket, is forgiven
1141                 * and we trampoline.  This protection is given regardless
1142                 * of whether we are 32/64 bit etc - if a distinction is
1143                 * required then define new on_trap protection types.
```



```
1144                 *
1145                 * On amd64, we can get a #gp from referencing addresses
1146                 * in the virtual address hole e.g. from a copyin or in
1147                 * update_sregs while updating user segment registers.
1148                 *
1149                 * On the 32-bit hypervisor we could also generate one in
1150                 * mfn_to_pfn by reaching around or into where the hypervisor
1151                 * lives which is protected by segmentation.
1152                 */

1154                /*
1155                 * If we're under on_trap() protection (see <sys/ontrap.h>),
1156                 * set ot_trap and trampoline back to the on_trap() call site
1157                 * for OT_DATA_ACCESS or OT_SEGMENT_ACCESS.
1158                 */
1159                if (ct->t_ontrap != NULL) {
1160                        int ttype =  ct->t_ontrap->ot_prot &
1161                            (OT_DATA_ACCESS | OT_SEGMENT_ACCESS);

1163                        if (ttype != 0) {
1164                                ct->t_ontrap->ot_trap |= ttype;
1165                                if (tudebug)
1166                                        showregs(type, rp, (caddr_t)0);
1167                                rp->r_pc = ct->t_ontrap->ot_trampoline;
1168                                goto cleanup;
1169                        }
1170                }

1172                /*
1173                 * If we're under lofault protection (copyin etc.),
1174                 * longjmp back to lofault with an EFAULT.
1175                 */
1176                if (ct->t_lofault) {
1177                        /*
1178                         * Fault is not resolvable, so just return to lofault
1179                         */
1180                        if (lodebug) {
1181                                showregs(type, rp, addr);
1182                                traceregs(rp);
1183                        }
1184                        rp->r_r0 = EFAULT;
1185                        rp->r_pc = ct->t_lofault;
1186                        goto cleanup;
1187                }

1189                /*
1190                 * We fall through to the next case, which repeats
1191                 * the OT_SEGMENT_ACCESS check which we've already
1192                 * done, so we'll always fall through to the
1193                 * T_STKFLT case.
1194                 */
1195                /*FALLTHROUGH*/
1196        case T_SEGFLT:  /* segment not present fault */
1197                /*
1198                 * One example of this is #NP in update_sregs while
1199                 * attempting to update a user segment register
1200                 * that points to a descriptor that is marked not
1201                 * present.
1202                 */
1203                if (ct->t_ontrap != NULL &&
1204                    ct->t_ontrap->ot_prot & OT_SEGMENT_ACCESS) {
1205                        ct->t_ontrap->ot_trap |= OT_SEGMENT_ACCESS;
1206                        if (tudebug)
1207                                showregs(type, rp, (caddr_t)0);
1208                        rp->r_pc = ct->t_ontrap->ot_trampoline;
1209                        goto cleanup;
```

```
1210                }
1211                /*FALLTHROUGH*/
1212        case T_STKFLT:  /* stack fault */
1213        case T_TSSFLT:  /* invalid TSS fault */
1214                if (tudebug)
1215                        showregs(type, rp, (caddr_t)0);
1216                if (kern_gpfault(rp))
1217                        (void) die(type, rp, addr, cpuid);
1218                goto cleanup;

1220        /*
1221         * ONLY 32-bit PROCESSES can USE a PRIVATE LDT! 64-bit apps
1222         * should have no need for them, so we put a stop to it here.
1223         *
1224         * So: not-present fault is ONLY valid for 32-bit processes with
1225         * a private LDT trying to do a system call. Emulate it.
1226         *
1227         * #gp fault is ONLY valid for 32-bit processes also, which DO NOT
1228         * have a private LDT, and are trying to do a system call. Emulate it.
1229         */

1231        case T_SEGFLT + USER:   /* segment not present fault */
1232        case T_GPFLT + USER:    /* general protection violation */
1233 #ifdef _SYSCALL32_IMPL
1234                if (p->p_model != DATAMODEL_NATIVE) {
1235 #endif /* _SYSCALL32_IMPL */
1236                if (instr_is_lcall_syscall((caddr_t)rp->r_pc)) {
1237                        if (type == T_SEGFLT + USER)
1238                                ASSERT(p->p_ldt != NULL);

1240                        if ((p->p_ldt == NULL && type == T_GPFLT + USER) ||
1241                            type == T_SEGFLT + USER) {

1243                        /*
1244                         * The user attempted a system call via the obsolete
1245                         * call gate mechanism. Because the process doesn't have
1246                         * an LDT (i.e. the ldtr contains 0), a #gp results.
1247                         * Emulate the syscall here, just as we do above for a
1248                         * #np trap.
1249                         */

1251                        /*
1252                         * Since this is a not-present trap, rp->r_pc points to
1253                         * the trapping lcall instruction. We need to bump it
1254                         * to the next insn so the app can continue on.
1255                         */
1256                        rp->r_pc += LCALLSIZE;
1257                        lwp->lwp_regs = rp;

1259                        /*
1260                         * Normally the microstate of the LWP is forced back to
1261                         * LMS_USER by the syscall handlers. Emulate that
1262                         * behavior here.
1263                         */
1264                        mstate = LMS_USER;

1266                        dosyscall();
1267                        goto out;
1268                        }
1269                }
1270 #ifdef _SYSCALL32_IMPL
1271                }
1272 #endif /* _SYSCALL32_IMPL */
1273                /*
1274                 * If the current process is using a private LDT and the
1275                 * trapping instruction is sysenter, the sysenter instruction
```



```
1276                 * has been disabled on the CPU because it destroys segment
1277                 * registers. If this is the case, rewrite the instruction to
1278                 * be a safe system call and retry it. If this occurs on a CPU
1279                 * which doesn't even support sysenter, the result of all of
1280                 * this will be to emulate that particular instruction.
1281                 */
1282                if (p->p_ldt != NULL &&
1283                    ldt_rewrite_syscall(rp, p, X86FSET_SEP))
1284                        goto out;

1286                /*FALLTHROUGH*/

1288        case T_BOUNDFLT + USER: /* bound fault */
1289        case T_STKFLT + USER:   /* stack fault */
1290        case T_TSSFLT + USER:   /* invalid TSS fault */
1291                if (tudebug)
1292                        showregs(type, rp, (caddr_t)0);
1293                siginfo.si_signo = SIGSEGV;
1294                siginfo.si_code  = SEGV_MAPERR;
1295                siginfo.si_addr  = (caddr_t)rp->r_pc;
1296                fault = FLTBOUNDS;
1297                break;

1299        case T_ALIGNMENT + USER:        /* user alignment error (486) */
1300                if (tudebug)
1301                        showregs(type, rp, (caddr_t)0);
1302                bzero(&siginfo, sizeof (siginfo));
1303                siginfo.si_signo = SIGBUS;
1304                siginfo.si_code = BUS_ADRALN;
1305                siginfo.si_addr = (caddr_t)rp->r_pc;
1306                fault = FLTACCESS;
1307                break;

1309        case T_SGLSTP + USER: /* single step/hw breakpoint exception */
1310                if (tudebug && tudebugbpt)
1311                        showregs(type, rp, (caddr_t)0);

1313                /* Was it single-stepping? */
1314                if (lwp->lwp_pcb.pcb_drstat & DR_SINGLESTEP) {
1315                        pcb_t *pcb = &lwp->lwp_pcb;

1317                        rp->r_ps &= ~PS_T;
1318                        /*
1319                         * If both NORMAL_STEP and WATCH_STEP are in effect,
1320                         * give precedence to WATCH_STEP.  If neither is set,
1321                         * user must have set the PS_T bit in %efl; treat this
1322                         * as NORMAL_STEP.
1323                         */
1324                        if ((fault = undo_watch_step(&siginfo)) == 0 &&
1325                            ((pcb->pcb_flags & NORMAL_STEP) ||
1326                            !(pcb->pcb_flags & WATCH_STEP))) {
1327                                siginfo.si_signo = SIGTRAP;
1328                                siginfo.si_code = TRAP_TRACE;
1329                                siginfo.si_addr = (caddr_t)rp->r_pc;
1330                                fault = FLTTRACE;
1331                        }
1332                        pcb->pcb_flags &= ~(NORMAL_STEP|WATCH_STEP);
1333                }
1334                break;

1336        case T_BPTFLT + USER:   /* breakpoint trap */
1337                if (tudebug && tudebugbpt)
1338                        showregs(type, rp, (caddr_t)0);
1339                /*
1340                 * int 3 (the breakpoint instruction) leaves the pc referring
1341                 * to the address one byte after the breakpointed address.
```

```
1342                 * If the P_PR_BPTADJ flag has been set via /proc, We adjust
1343                 * it back so it refers to the breakpointed address.
1344                 */
1345                if (p->p_proc_flag & P_PR_BPTADJ)
1346                        rp->r_pc--;
1347                siginfo.si_signo = SIGTRAP;
1348                siginfo.si_code  = TRAP_BRKPT;
1349                siginfo.si_addr  = (caddr_t)rp->r_pc;
1350                fault = FLTBPT;
1351                break;

1353        case T_AST:
1354                /*
1355                 * This occurs only after the cs register has been made to
1356                 * look like a kernel selector, either through debugging or
1357                 * possibly by functions like setcontext().  The thread is
1358                 * about to cause a general protection fault at common_iret()
1359                 * in locore.  We let that happen immediately instead of
1360                 * doing the T_AST processing.
1361                 */
1362                goto cleanup;

1364        case T_AST + USER:      /* profiling, resched, h/w error pseudo trap */
1365                if (lwp->lwp_pcb.pcb_flags & ASYNC_HWERR) {
1366                        proc_t *p = ttoproc(curthread);
1367                        extern void print_msg_hwerr(ctid_t ct_id, proc_t *p);

1369                        lwp->lwp_pcb.pcb_flags &= ~ASYNC_HWERR;
1370                        print_msg_hwerr(p->p_ct_process->conp_contract.ct_id,
1371                            p);
1372                        contract_process_hwerr(p->p_ct_process, p);
1373                        siginfo.si_signo = SIGKILL;
1374                        siginfo.si_code = SI_NOINFO;
1375                } else if (lwp->lwp_pcb.pcb_flags & CPC_OVERFLOW) {
1376                        lwp->lwp_pcb.pcb_flags &= ~CPC_OVERFLOW;
1377                        if (kcpc_overflow_ast()) {
1378                                /*
1379                                 * Signal performance counter overflow
1380                                 */
1381                                if (tudebug)
1382                                        showregs(type, rp, (caddr_t)0);
1383                                bzero(&siginfo, sizeof (siginfo));
1384                                siginfo.si_signo = SIGEMT;
1385                                siginfo.si_code = EMT_CPCOVF;
1386                                siginfo.si_addr = (caddr_t)rp->r_pc;
1387                                fault = FLTCPCOVF;
1388                        }
1389                }

1391                break;
1392        }

1394        /*
1395         * We can't get here from a system trap
1396         */
1397        ASSERT(type & USER);

1399        if (fault) {
1400                /* We took a fault so abort single step. */
1401                lwp->lwp_pcb.pcb_flags &= ~(NORMAL_STEP|WATCH_STEP);
1402                /*
1403                 * Remember the fault and fault adddress
1404                 * for real-time (SIGPROF) profiling.
1405                 */
1406                lwp->lwp_lastfault = fault;
1407                lwp->lwp_lastfaddr = siginfo.si_addr;
```



```
1409                DTRACE_PROC2(fault, int, fault, ksiginfo_t *, &siginfo);

1411                /*
1412                 * If a debugger has declared this fault to be an
1413                 * event of interest, stop the lwp.  Otherwise just
1414                 * deliver the associated signal.
1415                 */
1416                if (siginfo.si_signo != SIGKILL &&
1417                    prismember(&p->p_fltmask, fault) &&
1418                    stop_on_fault(fault, &siginfo) == 0)
1419                        siginfo.si_signo = 0;
1420        }

1422        if (siginfo.si_signo)
1423                trapsig(&siginfo, (fault != FLTFPE && fault != FLTCPCOVF));

1425        if (lwp->lwp_oweupc)
1426                profil_tick(rp->r_pc);

1428        if (ct->t_astflag | ct->t_sig_check) {
1429                /*
1430                 * Turn off the AST flag before checking all the conditions that
1431                 * may have caused an AST.  This flag is on whenever a signal or
1432                 * unusual condition should be handled after the next trap or
1433                 * syscall.
1434                 */
1435                astoff(ct);
1436                /*
1437                 * If a single-step trap occurred on a syscall (see above)
1438                 * recognize it now.  Do this before checking for signals
1439                 * because deferred_singlestep_trap() may generate a SIGTRAP to
1440                 * the LWP or may otherwise mark the LWP to call issig(FORREAL).
1441                 */
1442                if (lwp->lwp_pcb.pcb_flags & DEBUG_PENDING)
1443                        deferred_singlestep_trap((caddr_t)rp->r_pc);

1445                ct->t_sig_check = 0;

1447                /*
1448                 * As in other code paths that check against TP_CHANGEBIND,
1449                 * we perform the check first without p_lock held -- only
1450                 * acquiring p_lock in the unlikely event that it is indeed
1451                 * set.  This is safe because we are doing this after the
1452                 * astoff(); if we are racing another thread setting
1453                 * TP_CHANGEBIND on us, we will pick it up on a subsequent
1454                 * lap through.
1455                 */
1456                if (curthread->t_proc_flag & TP_CHANGEBIND) {
1457                        mutex_enter(&p->p_lock);
1458                        if (curthread->t_proc_flag & TP_CHANGEBIND) {
1459                                timer_lwpbind();
1460                                curthread->t_proc_flag &= ~TP_CHANGEBIND;
1461                        }
1462                        mutex_exit(&p->p_lock);
1463                }

1465                /*
1466                 * for kaio requests that are on the per-process poll queue,
1467                 * aiop->aio_pollq, they're AIO_POLL bit is set, the kernel
1468                 * should copyout their result_t to user memory. by copying
1469                 * out the result_t, the user can poll on memory waiting
1470                 * for the kaio request to complete.
1471                 */
1472                if (p->p_aio)
1473                        aio_cleanup(0);
```

```
1474                 /*
1475                  * If this LWP was asked to hold, call holdlwp(), which will
1476                  * stop.  holdlwps() sets this up and calls pokelwps() which
1477                  * sets the AST flag.
1478                  *
1479                  * Also check TP_EXITLWP, since this is used by fresh new LWPs
1480                  * through lwp_rtt().  That flag is set if the lwp_create(2)
1481                  * syscall failed after creating the LWP.
1482                  */
1483                 if (ISHOLD(p))
1484                         holdlwp();

1486                 /*
1487                  * All code that sets signals and makes ISSIG evaluate true must
1488                  * set t_astflag afterwards.
1489                  */
1490                 if (ISSIG_PENDING(ct, lwp, p)) {
1491                         if (issig(FORREAL))
1492                                 psig();
1493                         ct->t_sig_check = 1;
1494                 }

1496                 if (ct->t_rprof != NULL) {
1497                         realsigprof(0, 0, 0);
1498                         ct->t_sig_check = 1;
1499                 }

1501                 /*
1502                  * /proc can't enable/disable the trace bit itself
1503                  * because that could race with the call gate used by
1504                  * system calls via "lcall". If that happened, an
1505                  * invalid EFLAGS would result. prstep()/prnostep()
1506                  * therefore schedule an AST for the purpose.
1507                  */
1508                 if (lwp->lwp_pcb.pcb_flags & REQUEST_STEP) {
1509                         lwp->lwp_pcb.pcb_flags &= ~REQUEST_STEP;
1510                         rp->r_ps |= PS_T;
1511                 }
1512                 if (lwp->lwp_pcb.pcb_flags & REQUEST_NOSTEP) {
1513                         lwp->lwp_pcb.pcb_flags &= ~REQUEST_NOSTEP;
1514                         rp->r_ps &= ~PS_T;
1515                 }
1516         }

1518 out:    /* We can't get here from a system trap */
1519         ASSERT(type & USER);

1521         if (ISHOLD(p))
1522                 holdlwp();

1524         /*
1525          * Set state to LWP_USER here so preempt won't give us a kernel
1526          * priority if it occurs after this point.  Call CL_TRAPRET() to
1527          * restore the user-level priority.
1528          *
1529          * It is important that no locks (other than spinlocks) be entered
1530          * after this point before returning to user mode (unless lwp_state
1531          * is set back to LWP_SYS).
1532          */
1533         lwp->lwp_state = LWP_USER;

1535         if (ct->t_trapret) {
1536                 ct->t_trapret = 0;
1537                 thread_lock(ct);
1538                 CL_TRAPRET(ct);
1539                 thread_unlock(ct);
```



```
1540         }
1541         if (CPU->cpu_runrun || curthread->t_schedflag & TS_ANYWAITQ)
1542                 preempt();
1543         prunstop();
1544         (void) new_mstate(ct, mstate);

1546         /* Kernel probe */
1547         TNF_PROBE_1(thread_state, "thread", /* CSTYLED */,
1548             tnf_microstate, state, LMS_USER);

1550         return;

1552 cleanup:        /* system traps end up here */
1553         ASSERT(!(type & USER));
1554 }
_____unchanged_portion_omitted_

1668 /*
1669  * Print out debugging info.
1670  */
1671 static void
1672 showregs(uint_t type, struct regs *rp, caddr_t addr)
1673 {
1674         int s;

1676         s = spl7();
1677         type &= ~USER;
1678         if (PTOU(curproc)->u_comm[0])
1679                 printf("%s: ", PTOU(curproc)->u_comm);
1680         if (type < TRAP_TYPES)
1681                 printf("#%s %s\n", trap_type_mnemonic[type], trap_type[type]);
1682         else
1683                 switch (type) {
1684                 case T_SYSCALL:
1685                         printf("Syscall Trap:\n");
1686                         break;
1687                 case T_AST:
1688                         printf("AST\n");
1689                         break;
1690                 default:
1691                         printf("Bad Trap = %d\n", type);
1692                         break;
1693                 }
1694         if (type == T_PGFLT) {
1695                 printf("Bad %s fault at addr=0x%lx\n",
1696                     USERMODE(rp->r_cs) ? "user": "kernel", (uintptr_t)addr);
1697         } else if (addr) {
1698                 printf("addr=0x%lx\n", (uintptr_t)addr);
1699         }

1701         printf("pid=%d, pc=0x%lx, sp=0x%lx, eflags=0x%lx\n",
1702             (ttoproc(curthread) && ttoproc(curthread)->p_pidp) ?
1703             ttoproc(curthread)->p_pid : 0, rp->r_pc, rp->r_sp, rp->r_ps);

1705 #if defined(__lint)
1706         /*
1707          * this clause can be deleted when lint bug 4870403 is fixed
1708          * (lint thinks that bit 32 is illegal in a %b format string)
1709          */
1710         printf("cr0: %x  cr4: %b\n",
1711             (uint_t)getcr0(), (uint_t)getcr4(), FMT_CR4);
1712 #else
1713         printf("cr0: %b  cr4: %b\n",
1714             (uint_t)getcr0(), FMT_CR0, (uint_t)getcr4(), FMT_CR4);
1715 #endif  /* __lint */
```

```
1717          printf("cr2: %lx  ", getcr2());
1741          printf("cr2: %lx", getcr2());
1718 #if !defined(__xpv)
1719          printf("cr3: %lx  ", getcr3());
1743          printf("cr3: %lx", getcr3());
1720 #if defined(__amd64)
1721          printf("cr8: %lx\n", getcr8());
1722 #endif
1723 #endif
1724          printf("\n");

1726          dumpregs(rp);
1727          splx(s);
1728 }
______unchanged_portion_omitted_

1822 #endif  /* __i386 */

1824 /*
1825  * Test to see if the instruction is part of _sys_rtt (or the KPTI trampolines
1826  * which are used by _sys_rtt).
1849  * Test to see if the instruction is part of _sys_rtt.
1827  *
1828  * Again on the hypervisor if we try to IRET to user land with a bad code
1829  * or stack selector we will get vectored through xen_failsafe_callback.
1830  * In which case we assume we got here via _sys_rtt since we only allow
1831  * IRET to user land to take place in _sys_rtt.
1832  */
1833 static int
1834 instr_is_sys_rtt(caddr_t pc)
1835 {
1836          extern void _sys_rtt(), _sys_rtt_end();

1838 #if !defined(__xpv)
1839          extern void tr_sysc_ret_start(), tr_sysc_ret_end();
1840          extern void tr_intr_ret_start(), tr_intr_ret_end();

1842          if ((uintptr_t)pc >= (uintptr_t)tr_sysc_ret_start &&
1843              (uintptr_t)pc <= (uintptr_t)tr_sysc_ret_end)
1844                  return (1);

1846          if ((uintptr_t)pc >= (uintptr_t)tr_intr_ret_start &&
1847              (uintptr_t)pc <= (uintptr_t)tr_intr_ret_end)
1848                  return (1);
1849 #endif

1851          if ((uintptr_t)pc < (uintptr_t)_sys_rtt ||
1852              (uintptr_t)pc > (uintptr_t)_sys_rtt_end)
1853                  return (0);

1855          return (1);
1856 }
______unchanged_portion_omitted_
```

```
**********************************************************
    5476 Fri Apr  6 17:25:03 2018
new/usr/src/uts/i86pc/sys/mach_mmu.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 _SYS_MACH_MMU_H
#define _SYS_MACH_MMU_H

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

#ifdef __cplusplus
extern "C" {
#endif

#ifndef _ASM

#include <sys/types.h>
#include <sys/systm.h>

/*
 * Platform-dependent MMU routines and types.
 *
 * WARNING: this header file is used by both dboot and i86pc, so don't go using
 * normal kernel headers.
 */

#define TWO_MEG         (2 * 1024 * 1024)

/*
 * This is:
 *      The kernel nucleus pagesizes, ie: bi->bi_kseg_size
 *      The grub 64 bit file load address (see multiboot header in dboot_grub.s)
 *      The grub 32 bit and hypervisor physical load addresses of
 *          the kernel text/data (see Mapfile.unix)
 */
#define FOUR_MEG        (4 * 1024 * 1024)
```



```
#define ONE_GIG         (1024 * 1024 * 1024)
#define FOUR_GIG        ((uint64_t)4 * ONE_GIG)

#define MMU_STD_PAGESIZE        4096
#ifdef __amd64
#define MMU_STD_PAGEMASK        0xFFFFFFFFFFFFF000ULL
#else
#define MMU_STD_PAGEMASK        0xFFFFF000UL
#endif

/*
 * Defines for the bits in X86 and AMD64 Page Tables
 *
 * Notes:
 *
 * Largepages and PAT bits:
 *
 * bit 7 at level 0 is the PAT bit
 * bit 7 above level 0 is the Pagesize bit (set for large page)
 * bit 12 (when a large page) is the PAT bit
 *
 * In Solaris the PAT/PWT/PCD values are set up so that:
 *
 * PAT & PWT -> Write Protected
 * PAT & PCD -> Write Combining
 * PAT by itself (PWT == 0 && PCD == 0) yields uncacheable (same as PCD == 1)
 *
 *
 * Permission bits:
 *
 * - PT_USER must be set in all levels for user pages
 * - PT_WRITE must be set in all levels for user writable pages
 * - PT_NX applies if set at any level
 *
 * For these, we use the "allow" settings in all tables above level 0 and only
 * ever disable things in PTEs.
 *
 * The use of PT_GLOBAL and PT_NX depend on being enabled in processor
 * control registers. Hence, we use a variable to reference these bit
 * masks. During hat_kern_setup() if the feature isn't enabled we
 * clear out the variables.
 */
#define PT_VALID        (0x001) /* a valid translation is present */
#define PT_WRITABLE     (0x002) /* the page is writable */
#define PT_USER         (0x004) /* the page is accessible by user mode */
#define PT_WRITETHRU    (0x008) /* write back caching is disabled (non-PAT) */
#define PT_NOCACHE      (0x010) /* page is not cacheable (non-PAT) */
#define PT_REF          (0x020) /* page was referenced */
#define PT_MOD          (0x040) /* page was modified */
#define PT_PAGESIZE     (0x080) /* above level 0, indicates a large page */
#define PT_PAT_4K       (0x080) /* at level 0, used for write combining */
#define PT_GLOBAL       (0x100) /* the mapping is global */
#define PT_SOFTWARE     (0xe00) /* software bits */

#define PT_PAT_LARGE    (0x1000)        /* PAT bit for large pages */

#define PT_PTPBITS      (PT_VALID | PT_USER | PT_WRITABLE | PT_REF)
#define PT_FLAGBITS     (0xfff) /* for masking off flag bits */

/*
 * The software bits are used by the HAT to track attributes.
 * Note that the attributes are inclusive as the values increase.
 *
 * PT_NOSYNC - The PT_REF/PT_MOD bits are not sync'd to page_t.
 *             The hat will install them as always set.
 *
```

```
 124  * PT_NOCONSIST - There is no hment entry for this mapping.
 125  *
 126  * PT_FOREIGN - used for the hypervisor, check via
 127  *              (pte & PT_SOFTWARE) >= PT_FOREIGN
 128  *              as it might set 0x800 for foreign grant table mappings.
 129  */
 130 #define PT_NOSYNC       (0x200) /* PTE was created with HAT_NOSYNC */
 131 #define PT_NOCONSIST    (0x400) /* PTE was created with HAT_LOAD_NOCONSIST */
 132 #define PT_FOREIGN      (0x600) /* MFN mapped on the hypervisor has no PFN */

 134 #ifndef _BOOT

 136 extern ulong_t getcr3(void);
 137 extern void setcr3(ulong_t);

 139 #define getcr3_pa() (getcr3() & MMU_PAGEMASK)
 140 #define getpcid() ((getcr4() & CR4_PCIDE) ? \
 141         (getcr3() & MMU_PAGEOFFSET) : PCID_NONE)

 143 extern void mmu_invlpg(caddr_t);

 145 #endif

 147 #ifdef __xpv
 148 #include <sys/xen_mmu.h>
 149 #else
 150 #include <sys/pc_mmu.h>
 151 #endif

 153 /*
 154  * The software extraction for a single Page Table Entry will always
 155  * be a 64 bit unsigned int. If running a non-PAE hat, the page table
 156  * access routines know to extend/shorten it to 32 bits.
 157  */
 158 typedef uint64_t x86pte_t;
 159 typedef uint32_t x86pte32_t;

 161 x86pte_t get_pteval(paddr_t, uint_t);
 162 void set_pteval(paddr_t, uint_t, uint_t, x86pte_t);
 163 paddr_t make_ptable(x86pte_t *, uint_t);
 164 x86pte_t *find_pte(uint64_t, paddr_t *, uint_t, uint_t);
 165 x86pte_t *map_pte(paddr_t, uint_t);

 154 #ifndef _BOOT
 155 ulong_t getcr3();
 156 #endif

 167 extern uint_t *shift_amt;
 168 extern uint_t ptes_per_table;
 169 extern paddr_t top_page_table;
 170 extern uint_t top_level;
 171 extern uint_t pte_size;
 172 extern uint_t shift_amt_nopae[];
 173 extern uint_t shift_amt_pae[];
 174 extern uint32_t lpagesize;

 176 #ifdef __cplusplus
 177 }
_____unchanged_portion_omitted_
```

```
**********************************************************
    7583 Fri Apr  6 17:25:03 2018
new/usr/src/uts/i86pc/sys/machcpuvar.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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.
 * Copyright 2011 Joyent, Inc. All rights reserved.
 */

#ifndef _SYS_MACHCPUVAR_H
#define _SYS_MACHCPUVAR_H

#ifdef  __cplusplus
extern "C" {
#endif

#include <sys/inttypes.h>
#include <sys/x_call.h>
#include <sys/tss.h>
#include <sys/segments.h>
#include <sys/rm_platter.h>
#include <sys/avintr.h>
#include <sys/pte.h>
#include <sys/stddef.h>
#include <sys/debug.h>
#include <sys/cpuvar.h>

#ifndef _ASM
/*
 * On a virtualized platform a virtual cpu may not be actually
 * on a physical cpu, especially in situations where a configuration has
 * more vcpus than pcpus.  This function tells us (if it's able) if the
 * specified vcpu is currently running on a pcpu.  Note if it is not
 * known or not able to determine, it will return the unknown state.
 */
#define VCPU_STATE_UNKNOWN      0
#define VCPU_ON_PCPU            1
#define VCPU_NOT_ON_PCPU        2
```



```
extern int vcpu_on_pcpu(processorid_t);

/*
 * Machine specific fields of the cpu struct
 * defined in common/sys/cpuvar.h.
 *
 * Note:  This is kinda kludgy but seems to be the best
 * of our alternatives.
 */

struct cpuid_info;
struct cpu_ucode_info;
struct cmi_hdl;

/*
 * A note about the hypervisor affinity bits: a one bit in the affinity mask
 * means the corresponding event channel is allowed to be serviced
 * by this cpu.
 */
struct xen_evt_data {
        ulong_t         pending_sel[PIL_MAX + 1]; /* event array selectors */
        ulong_t         pending_evts[PIL_MAX + 1][sizeof (ulong_t) * 8];
        ulong_t         evt_affinity[sizeof (ulong_t) * 8]; /* service on cpu */
};

struct kpti_frame {
        uint64_t        kf_lower_redzone;

        /* Stashed value of %cr3 when we entered the trampoline. */
        greg_t          kf_tr_cr3;

        /*
         * We use %r13-r14 as scratch registers in the trampoline code,
         * so stash those here "below" the rest of the stack so they can be
         * pushed/popped if needed.
         */
        greg_t          kf_r14;
        greg_t          kf_r13;

        /*
         * Part of this struct is used as the HW stack frame when taking an
         * interrupt on the user page table. The CPU is going to push a bunch
         * of regs onto the stack pointer set in the TSS/IDT (which we set to
         * &kf_rsp here).
         *
         * This is only a temporary holding area for them (we'll move them over
         * to the real interrupt stack once we've set %cr3).
         *
         * Note that these must be cleared during a process switch on this cpu.
         */
        greg_t          kf_err;         /* Bottom of initial hw stack frame */
        greg_t          kf_rip;
        greg_t          kf_cs;
        greg_t          kf_rflags;
        greg_t          kf_rsp;
        greg_t          kf_ss;

        greg_t          kf_tr_rsp;      /* Top of HW stack frame */
        /* We also write this with the %rsp value on tramp entry */

        /* Written to 0x1 when this kpti_frame is in use. */
        uint64_t        kf_tr_flag;

        uint64_t        kf_middle_redzone;

        /*
```

```
125          * The things we need to write to %cr3 to change between page tables.
126          * These live "above" the HW stack.
127          */
128         greg_t          kf_kernel_cr3;
129         greg_t          kf_user_cr3;
130         greg_t          kf_tr_ret_rsp;

132         uint64_t        kf_unused;              /* For 16-byte align */

134         uint64_t        kf_upper_redzone;
135 };

137 /*
138  * This first value, MACHCPU_SIZE is the size of all the members in the cpu_t
139  * AND struct machcpu, before we get to the mcpu_pad and the kpti area.
140  * The KPTI is used to contain per-CPU data that is visible in both sets of
141  * page-tables, and hence must be page-aligned and page-sized. See
142  * hat_pcp_setup().
143  *
144  * There is a CTASSERT in os/intr.c that checks these numbers.
145  */
146 #define MACHCPU_SIZE    (572 + 1584)
147 #define MACHCPU_PAD     (MMU_PAGESIZE - MACHCPU_SIZE)
148 #define MACHCPU_PAD2    (MMU_PAGESIZE - 16 - 3 * sizeof (struct kpti_frame))

150 struct  machcpu {
151         /*
152          * x_call fields - used for interprocessor cross calls
153          */
154         struct xc_msg   *xc_msgbox;
155         struct xc_msg   *xc_free;
156         xc_data_t       xc_data;
157         uint32_t        xc_wait_cnt;
158         volatile uint32_t xc_work_cnt;

160         int             mcpu_nodeid;            /* node-id */
161         int             mcpu_pri;               /* CPU priority */

163         struct hat      *mcpu_current_hat; /* cpu's current hat */

165         struct hat_cpu_info     *mcpu_hat_info;

167         volatile ulong_t        mcpu_tlb_info;

169         /* i86 hardware table addresses that cannot be shared */

171         user_desc_t     *mcpu_gdt;      /* GDT */
172         gate_desc_t     *mcpu_idt;      /* current IDT */

174         tss_t           *mcpu_tss;      /* TSS */
175         void            *mcpu_ldt;
176         size_t          mcpu_ldt_len;

178         kmutex_t        mcpu_ppaddr_mutex;
179         caddr_t         mcpu_caddr1;    /* per cpu CADDR1 */
180         caddr_t         mcpu_caddr2;    /* per cpu CADDR2 */
181         uint64_t        mcpu_caddr1pte;
182         uint64_t        mcpu_caddr2pte;

184         struct softint  mcpu_softinfo;
185         uint64_t        pil_high_start[HIGH_LEVELS];
186         uint64_t        intrstat[PIL_MAX + 1][2];

188         struct cpuid_info       *mcpu_cpi;

190 #if defined(__amd64)
```



```
191         greg_t  mcpu_rtmp_rsp;          /* syscall: temporary %rsp stash */
192         greg_t  mcpu_rtmp_r15;          /* syscall: temporary %r15 stash */
193 #endif

195         struct vcpu_info *mcpu_vcpu_info;
196         uint64_t        mcpu_gdtpa;     /* hypervisor: GDT physical address */

198         uint16_t mcpu_intr_pending;     /* hypervisor: pending intrpt levels */
199         uint16_t mcpu_ec_mbox;          /* hypervisor: evtchn_dev mailbox */
200         struct xen_evt_data *mcpu_evt_pend; /* hypervisor: pending events */

202         volatile uint32_t *mcpu_mwait;  /* MONITOR/MWAIT buffer */
203         void (*mcpu_idle_cpu)(void);    /* idle function */
204         uint16_t mcpu_idle_type;        /* CPU next idle type */
205         uint16_t max_cstates;           /* supported max cstates */

207         struct cpu_ucode_info   *mcpu_ucode_info;

209         void                    *mcpu_pm_mach_state;
210         struct cmi_hdl          *mcpu_cmi_hdl;
211         void                    *mcpu_mach_ctx_ptr;

213         /*
214          * A stamp that is unique per processor and changes
215          * whenever an interrupt happens. Userful for detecting
216          * if a section of code gets interrupted.
217          * The high order 16 bits will hold the cpu->cpu_id.
218          * The low order bits will be incremented on every interrupt.
219          */
220         volatile uint32_t       mcpu_istamp;

222         char                    mcpu_pad[MACHCPU_PAD];

224         /* This is the start of the page */
225         char                    mcpu_pad2[MACHCPU_PAD2];
226         struct kpti_frame       mcpu_kpti;
227         struct kpti_frame       mcpu_kpti_flt;
228         struct kpti_frame       mcpu_kpti_dbg;
229         char                    mcpu_pad3[16];
230 };

232 #define NINTR_THREADS   (LOCK_LEVEL-1)  /* number of interrupt threads */
233 #define MWAIT_HALTED    (1)             /* mcpu_mwait set when halting */
234 #define MWAIT_RUNNING   (0)             /* mcpu_mwait set to wakeup */
235 #define MWAIT_WAKEUP_IPI        (2)     /* need IPI to wakeup */
236 #define MWAIT_WAKEUP(cpu)       (*((cpu)->cpu_m.mcpu_mwait) = MWAIT_RUNNING)

238 #endif  /* _ASM */

240 /* Please DON'T add any more of this namespace-poisoning sewage here */

242 #define cpu_nodeid cpu_m.mcpu_nodeid
243 #define cpu_pri cpu_m.mcpu_pri
244 #define cpu_current_hat cpu_m.mcpu_current_hat
245 #define cpu_hat_info cpu_m.mcpu_hat_info
246 #define cpu_ppaddr_mutex cpu_m.mcpu_ppaddr_mutex
247 #define cpu_gdt cpu_m.mcpu_gdt
248 #define cpu_idt cpu_m.mcpu_idt
249 #define cpu_tss cpu_m.mcpu_tss
170 #define cpu_ldt cpu_m.mcpu_ldt
250 #define cpu_caddr1 cpu_m.mcpu_caddr1
251 #define cpu_caddr2 cpu_m.mcpu_caddr2
252 #define cpu_softinfo cpu_m.mcpu_softinfo
253 #define cpu_caddr1pte cpu_m.mcpu_caddr1pte
254 #define cpu_caddr2pte cpu_m.mcpu_caddr2pte
```

```
 256 #ifdef  __cplusplus
 257 }
_____unchanged_portion_omitted_
```

```
**********************************************************
    9789 Fri Apr  6 17:25:04 2018
new/usr/src/uts/i86pc/sys/machparam.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2015 by Delphix. All rights reserved.
 * Copyright 2018 Joyent, Inc.
 * Copyright 2016 Joyent, Inc.
 */

/*      Copyright (c) 1988 AT&T */
/*        All Rights Reserved   */


#ifndef _SYS_MACHPARAM_H
#define _SYS_MACHPARAM_H

#if !defined(_ASM)
#include <sys/types.h>

#if defined(__xpv)
#include <sys/xpv_impl.h>
#endif

#endif

#ifdef  __cplusplus
extern "C" {
#endif

#ifndef _ASM
#define ADDRESS_C(c)    c ## ul
#else   /* _ASM */
#define ADDRESS_C(c)    (c)
#endif  /* _ASM */

/*
 * Machine dependent parameters and limits.
 */

#if defined(__amd64)
/*
```



```
 * If NCPU grows beyond 256, sizing for the x86 comm page will require
 * adjustment.
 */
#define NCPU    256
#define NCPU_LOG2       8
#elif defined(__i386)
#define NCPU    32
#define NCPU_LOG2       5
#endif

/* NCPU_P2 is NCPU rounded to a power of 2 */
#define NCPU_P2 (1 << NCPU_LOG2)

/*
 * The value defined below could grow to 16. hat structure and
 * page_t have room for 16 nodes.
 */
#define MAXNODES        4
#define NUMA_NODEMASK   0x0f

/*
 * Define the FPU symbol if we could run on a machine with an external
 * FPU (i.e. not integrated with the normal machine state like the vax).
 *
 * The fpu is defined in the architecture manual, and the kernel hides
 * its absence if it is not present, that's pretty integrated, no?
 */

/* supported page sizes */
#define MMU_PAGE_SIZES  3

/*
 * MMU_PAGES* describes the physical page size used by the mapping hardware.
 * PAGES* describes the logical page size used by the system.
 */

#define MMU_PAGESIZE    0x1000          /* 4096 bytes */
#define MMU_PAGESHIFT   12              /* log2(MMU_PAGESIZE) */

#if !defined(_ASM)
#define MMU_PAGEOFFSET  (MMU_PAGESIZE-1) /* Mask of address bits in page */
#else   /* _ASM */
#define MMU_PAGEOFFSET  _CONST(MMU_PAGESIZE-1)  /* assembler lameness */
#endif  /* _ASM */

#define MMU_PAGEMASK    (~MMU_PAGEOFFSET)

#define PAGESIZE        0x1000          /* All of the above, for logical */
#define PAGESHIFT       12
#define PAGEOFFSET      (PAGESIZE - 1)
#define PAGEMASK        (~PAGEOFFSET)

/*
 * DATA_ALIGN is used to define the alignment of the Unix data segment.
 */
#define DATA_ALIGN      PAGESIZE

/*
 * DEFAULT KERNEL THREAD stack size (in pages).
 */
#if defined(__amd64)
#define DEFAULTSTKSZ_NPGS       5
#elif defined(__i386)
#define DEFAULTSTKSZ_NPGS       3
#endif
```

```
125 #if !defined(_ASM)
126 #define DEFAULTSTKSZ    (DEFAULTSTKSZ_NPGS * PAGESIZE)
127 #else   /* !_ASM */
128 #define DEFAULTSTKSZ    _MUL(DEFAULTSTKSZ_NPGS, PAGESIZE) /* as(1) lameness */
129 #endif  /* !_ASM */

131 /*
132  * KERNELBASE is the virtual address at which the kernel segments start in
133  * all contexts.
134  *
135  * KERNELBASE is not fixed.  The value of KERNELBASE can change with
136  * installed memory or on 32 bit systems the eprom variable 'eprom_kernelbase'.
137  *
138  * common/conf/param.c requires a compile time defined value for KERNELBASE.
139  * This value is save in the variable _kernelbase.  _kernelbase may then be
140  * modified with to a different value in i86pc/os/startup.c.
141  *
142  * Most code should be using kernelbase, which resolves to a reference to
143  * _kernelbase.
144  */
145 #define KERNEL_TEXT_amd64       UINT64_C(0xfffffffffb800000)

147 #ifdef __i386

149 #define KERNEL_TEXT_i386        ADDRESS_C(0xfe800000)

151 /*
152  * We don't use HYPERVISOR_VIRT_START, as we need both the PAE and non-PAE
153  * versions in our code. We always compile based on the lower PAE address.
154  */
155 #define KERNEL_TEXT_i386_xpv    \
156         (HYPERVISOR_VIRT_START_PAE - 3 * ADDRESS_C(0x400000))

158 #endif /* __i386 */

160 #if defined(__amd64)

162 #define KERNELBASE      ADDRESS_C(0xfffffd8000000000)

164 /*
165  * Size of the unmapped "red zone" at the very bottom of the kernel's
166  * address space.  Corresponds to 1 slot in the toplevel pagetable.
167  */
168 #define KERNEL_REDZONE_SIZE   ((uintptr_t)1 << 39)

170 /*
171  * Base of 'core' heap area, which is used for kernel and module text/data
172  * that must be within a 2GB range to allow for rip-relative addressing.
173  */
174 #define COREHEAP_BASE   ADDRESS_C(0xfffffffffc0000000)

176 /*
177  * Beginning of the segkpm window. A lower value than this is used if
178  * physical addresses exceed 1TB. See i86pc/os/startup.c
179  */
180 #define SEGKPM_BASE     ADDRESS_C(0xffffffe0000000000)

182 /*
183  * This is valloc_base, above seg_kpm, but below everything else.
184  * A lower value than this may be used if SEGKPM_BASE is adjusted.
185  * See i86pc/os/startup.c
186  */
187 #define VALLOC_BASE     ADDRESS_C(0xffffff0000000000)

189 /*
190  * default and boundary sizes for segkp
```



```
191  */
192 #define SEGKPDEFSIZE    (2L * 1024L * 1024L * 1024L)            /*   2G */
193 #define SEGKPMAXSIZE    (8L * 1024L * 1024L * 1024L)            /*   8G */
194 #define SEGKPMINSIZE    (200L * 1024 * 1024L)                   /* 200M */

196 /*
197  * minimum size for segzio
198  */
199 #define SEGZIOMINSIZE   (400L * 1024 * 1024L)                   /* 400M */

201 /*
202  * During intial boot we limit heap to the top 4Gig.
203  */
204 #define BOOT_KERNELHEAP_BASE    ADDRESS_C(0xffffffff00000000)

206 /*
207  * VMWare works best if we don't use the top 64Meg of memory for amd64.
208  * Set KERNEL_TEXT to top_o_memory - 64Meg - 8 Meg for 8Meg of nucleus pages.
209  */
210 #define PROMSTART       ADDRESS_C(0xffc00000)
211 #define KERNEL_TEXT     KERNEL_TEXT_amd64

213 /*
214  * Virtual address range available to the debugger
215  */
216 #define SEGDEBUGBASE    ADDRESS_C(0xfffffffff800000)
217 #define SEGDEBUGSIZE    ADDRESS_C(0x400000)

219 /*
220  * Define upper limit on user address space
221  *
222  * In amd64, the upper limit on a 64-bit user address space is 1 large page
223  * (2MB) below kernelbase.  The upper limit for a 32-bit user address space
224  * is 1 small page (4KB) below the top of the 32-bit range.  The 64-bit
225  * limit give dtrace the red zone it needs below kernelbase.  The 32-bit
226  * limit gives us a small red zone to detect address-space overruns in a
227  * user program.
228  *
229  * On the hypervisor, we limit the user to memory below the VA hole.
230  * Subtract 1 large page for a red zone.
231  */
232 #if defined(__xpv)
233 #define USERLIMIT       ADDRESS_C(0x00007fffffe00000)
234 #else
235 #define USERLIMIT       ADDRESS_C(0xfffffd7fffe00000)
236 #endif

238 #ifdef bug_5074717_is_fixed
239 #define USERLIMIT32     ADDRESS_C(0xffffff000)
240 #else
241 #define USERLIMIT32     ADDRESS_C(0xfefff000)
242 #endif

244 #elif defined(__i386)

246 #ifdef DEBUG
247 #define KERNELBASE      ADDRESS_C(0xc8000000)
248 #else
249 #define KERNELBASE      ADDRESS_C(0xd4000000)
250 #endif

252 #define KERNELBASE_MAX  ADDRESS_C(0xe0000000)

254 /*
255  * The i386 ABI requires that the user address space be at least 3Gb
256  * in size.  KERNELBASE_ABI_MIN is used as the default KERNELBASE for
```

```
 257  * physical memory configurations > 4gb.
 258  */
 259 #define KERNELBASE_ABI_MIN      ADDRESS_C(0xc0000000)

 261 /*
 262  * Size of the unmapped "red zone" at the very bottom of the kernel's
 263  * address space.  Since segmap start immediately above the red zone, this
 264  * needs to be MAXBSIZE aligned.
 265  */
 266 #define KERNEL_REDZONE_SIZE   MAXBSIZE

 268 /*
 269  * This is the last 4MB of the 4G address space. Some psm modules
 270  * need this region of virtual address space mapped 1-1
 271  * The top 64MB of the address space is reserved for the hypervisor.
 272  */
 273 #define PROMSTART       ADDRESS_C(0xffc00000)
 274 #ifdef __xpv
 275 #define KERNEL_TEXT     KERNEL_TEXT_i386_xpv
 276 #else
 277 #define KERNEL_TEXT     KERNEL_TEXT_i386
 278 #endif

 280 /*
 281  * Virtual address range available to the debugger
 282  * We place it just above the kernel text (4M) and kernel data (4M).
 283  */
 284 #define SEGDEBUGBASE    (KERNEL_TEXT + ADDRESS_C(0x800000))
 285 #define SEGDEBUGSIZE    ADDRESS_C(0x400000)

 287 /*
 288  * Define upper limit on user address space
 289  */
 290 #define USERLIMIT       KERNELBASE
 291 #define USERLIMIT32     USERLIMIT

 293 #endif  /* __i386 */

 295 /*
 296  * Reserve pages just below KERNEL_TEXT for the GDT, IDT, LDT, TSS and debug
 297  * info.
 296  * Reserve pages just below KERNEL_TEXT for the GDT, IDT, TSS and debug info.
 298  *
 299  * For now, DEBUG_INFO_VA must be first in this list for "xm" initiated dumps
 300  * of solaris domUs to be usable with mdb. Relying on a fixed VA is not viable
 301  * long term, but it's the best we've got for now.
 302  */
 303 #if !defined(_ASM)
 304 #define DEBUG_INFO_VA   (KERNEL_TEXT - MMU_PAGESIZE)
 305 #define GDT_VA          (DEBUG_INFO_VA - MMU_PAGESIZE)
 306 #define IDT_VA          (GDT_VA - MMU_PAGESIZE)
 307 #define LDT_VA          (IDT_VA - (16 * MMU_PAGESIZE))
 308 #define KTSS_VA         (LDT_VA - MMU_PAGESIZE)
 306 #define KTSS_VA         (IDT_VA - MMU_PAGESIZE)
 309 #define DFTSS_VA        (KTSS_VA - MMU_PAGESIZE)
 310 #define MISC_VA_BASE    (DFTSS_VA)
 311 #define MISC_VA_SIZE    (KERNEL_TEXT - MISC_VA_BASE)
 312 #endif /* !_ASM */

 314 #if !defined(_ASM) && !defined(_KMDB)
 315 extern uintptr_t kernelbase, segmap_start, segmapsize;
 316 #endif

 318 /*
 319  * ARGSBASE is the base virtual address of the range which
 320  * the kernel uses to map the arguments for exec.
```



```
 321  */
 322 #define ARGSBASE        PROMSTART

 324 /*
 325  * reserve space for modules
 326  */
 327 #define MODTEXT (1024 * 1024 * 2)
 328 #define MODDATA (1024 * 300)

 330 /*
 331  * The heap has a region allocated from it of HEAPTEXT_SIZE bytes specifically
 332  * for module text.
 333  */
 334 #define HEAPTEXT_SIZE           (64 * 1024 * 1024)      /* bytes */

 336 /*
 337  * Size of a kernel threads stack.  It must be a whole number of pages
 338  * since the segment it comes from will only allocate space in pages.
 339  */
 340 #define T_STACKSZ       2*PAGESIZE

 342 /*
 343  * Size of a cpu startup thread stack.  (It must be a whole number of pages
 344  * since the containing segment only allocates space in pages.)
 345  */

 347 #define STARTUP_STKSZ   3*PAGESIZE

 349 /*
 350  * Bus types
 351  */
 352 #define BTISA           1
 353 #define BTEISA          2
 354 #define BTMCA           3

 356 #ifdef  __cplusplus
 357 }
_____unchanged_portion_omitted_
```

```
**********************************************************
    3763 Fri Apr  6 17:25:04 2018
new/usr/src/uts/i86pc/sys/machprivregs.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 _SYS_MACHPRIVREGS_H
#define _SYS_MACHPRIVREGS_H

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

/*
 * Platform dependent instruction sequences for manipulating
 * privileged state
 */

#ifdef __cplusplus
extern "C" {
#endif

#define ASSERT_UPCALL_MASK_IS_SET               /* empty */

/*
 * CLI and STI
 */

#define CLI(r)                  \
        cli

#define STI                     \
        sti

/*
 * Used to re-enable interrupts in the body of exception handlers
 */

#if defined(__amd64)
```



```
#define ENABLE_INTR_FLAGS               \
        pushq   $F_ON;                  \
        popfq

#elif defined(__i386)

#define ENABLE_INTR_FLAGS               \
        pushl   $F_ON;                  \
        popfl

#endif  /* __i386 */

/*
 * IRET and SWAPGS
 */
#if defined(__amd64)

#define IRET    iretq
#define SYSRETQ sysretq
#define SYSRETL sysretl
#define SWAPGS  swapgs
#define XPV_TRAP_POP    /* empty */
#define XPV_TRAP_PUSH   /* empty */

#elif defined(__i386)

#define IRET    iret

#endif  /* __i386 */

#define XPV_TRAP_POP    /* empty */
#define XPV_TRAP_PUSH   /* empty */
#define CLEAN_CS        /* empty */


/*
 * Macros for saving the original segment registers and restoring them
 * for fast traps.
 */
#if defined(__amd64)

/*
 * Smaller versions of INTR_PUSH and INTR_POP for fast traps.
 * The following registers have been pushed onto the stack by
 * hardware at this point:
 *
 *      greg_t  r_rip;
 *      greg_t  r_cs;
 *      greg_t  r_rfl;
 *      greg_t  r_rsp;
 *      greg_t  r_ss;
 *
 * This handler is executed both by 32-bit and 64-bit applications.
 * 64-bit applications allow us to treat the set (%rdi, %rsi, %rdx,
 * %rcx, %r8, %r9, %r10, %r11, %rax) as volatile across function calls.
 * However, 32-bit applications only expect (%eax, %edx, %ecx) to be volatile
 * across a function call -- in particular, %esi and %edi MUST be saved!
 *
 * We could do this differently by making a FAST_INTR_PUSH32 for 32-bit
 * programs, and FAST_INTR_PUSH for 64-bit programs, but it doesn't seem
 * particularly worth it.
 */
#define FAST_INTR_PUSH                  \
        INTGATE_INIT_KERNEL_FLAGS;      \
        subq    $REGOFF_RIP, %rsp;      \
```

```
 122         movq    %rsi, REGOFF_RSI(%rsp); \
 123         movq    %rdi, REGOFF_RDI(%rsp); \
 124         swapgs

 126 #define FAST_INTR_POP                   \
 127         swapgs;                         \
 128         movq    REGOFF_RSI(%rsp), %rsi; \
 129         movq    REGOFF_RDI(%rsp), %rdi; \
 130         addq    $REGOFF_RIP, %rsp

 132 #define FAST_INTR_RETURN        jmp tr_iret_user
 132 #define FAST_INTR_RETURN        iretq

 134 #elif defined(__i386)

 136 #define FAST_INTR_PUSH          \
 137         cld;                    \
 138         __SEGREGS_PUSH          \
 139         __SEGREGS_LOAD_KERNEL

 141 #define FAST_INTR_POP           \
 142         __SEGREGS_POP

 144 #define FAST_INTR_RETURN        iret

 146 #endif  /* __i386 */

 148 /*
 149  * Handling the CR0.TS bit for floating point handling.
 150  *
 151  * When the TS bit is *set*, attempts to touch the floating
 152  * point hardware will result in a #nm trap.
 153  */
 154 #if defined(__amd64)

 156 #define STTS(rtmp)              \
 157         movq    %cr0, rtmp;     \
 158         orq     $CR0_TS, rtmp;  \
 159         movq    rtmp, %cr0

 161 #elif defined(__i386)

 163 #define STTS(rtmp)              \
 164         movl    %cr0, rtmp;     \
 165         orl     $CR0_TS, rtmp;  \
 166         movl    rtmp, %cr0

 168 #endif  /* __i386 */

 170 #define CLTS                    \
 171         clts

 173 #ifdef __cplusplus
 174 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    1833 Fri Apr  6 17:25:04 2018
new/usr/src/uts/i86pc/sys/pc_mmu.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 _SYS_PC_MMU_H
#define _SYS_PC_MMU_H

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

#ifdef __cplusplus
extern "C" {
#endif

/*
 * Platform-dependent MMU routines and types for real x86 hardware.
 *
 * WARNING: this header file is used by both dboot and i86pc, so don't go using
 * normal kernel headers.
 */

#define IN_HYPERVISOR_VA(va) (__lintzero)

void reload_cr3(void);

#define pa_to_ma(pa) (pa)
#define ma_to_pa(ma) (ma)
#define pfn_to_mfn(pfn) (pfn)
#define mfn_to_pfn(mfn) (mfn)

#ifndef _BOOT

extern uint64_t kpti_safe_cr3;
void mmu_tlbflush_entry(caddr_t);
void setcr3(ulong_t);

#define INVPCID_ADDR            (0)
```



```
#define INVPCID_ID              (1)
#define INVPCID_ALL_GLOBAL      (2)
#define INVPCID_ALL_NONGLOBAL   (3)

extern void invpcid_insn(uint64_t, uint64_t, uintptr_t);
extern void tr_mmu_flush_user_range(uint64_t, size_t, size_t, uint64_t);

#if defined(__GNUC__)
#include <asm/mmu.h>
#endif

#endif /* !_BOOT */

#ifdef __cplusplus
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    4107 Fri Apr  6 17:25:04 2018
new/usr/src/uts/i86pc/sys/rm_platter.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */
/*
 * Copyright 2018 Joyent, Inc.
 * Copyright 2011 Joyent, Inc. All rights reserved.
 */

#ifndef _SYS_RM_PLATTER_H
#define _SYS_RM_PLATTER_H

#include <sys/types.h>
#include <sys/tss.h>
#include <sys/segments.h>

#ifdef  __cplusplus
extern "C" {
#endif

#define RM_PLATTER_CODE_SIZE            0x400
#define RM_PLATTER_CPU_HALT_CODE_SIZE   0x100

typedef struct rm_platter {
        char            rm_code[RM_PLATTER_CODE_SIZE];
        char            rm_cpu_halt_code[RM_PLATTER_CPU_HALT_CODE_SIZE];
#if defined(__amd64)
        /*
         * The compiler will want to 64-bit align the 64-bit rm_gdt_base
         * pointer, so we need to add an extra four bytes of padding here to
         * make sure rm_gdt_lim and rm_gdt_base will align to create a proper
         * ten byte GDT pseudo-descriptor.
         */
        uint32_t        rm_gdt_pad;
#endif  /* __amd64 */
        ushort_t        rm_debug;
```



```
        ushort_t        rm_gdt_lim;     /* stuff for lgdt */
        user_desc_t     *rm_gdt_base;
#if defined(__amd64)
        /*
         * The compiler will want to 64-bit align the 64-bit rm_idt_base
         * pointer, so we need to add an extra four bytes of padding here to
         * make sure rm_idt_lim and rm_idt_base will align to create a proper
         * ten byte IDT pseudo-descriptor.
         */
        uint32_t        rm_idt_pad;
#endif  /* __amd64 */
        ushort_t        rm_cpu_halted;  /* non-zero if CPU has been halted */
        ushort_t        rm_idt_lim;     /* stuff for lidt */
        gate_desc_t     *rm_idt_base;
        uint_t          rm_pdbr;        /* cr3 value */
        uint_t          rm_cpu;         /* easy way to know which CPU we are */
        uint_t          rm_filler3;
        uint_t          rm_cr4;         /* cr4 value on cpu0 */
#if defined(__amd64)
        /*
         * Temporary GDT for the brief transition from real mode to protected
         * mode before a CPU continues on into long mode.
         *
         * Putting it here assures it will be located in identity mapped memory
         * (va == pa, 1:1).
         *
         * rm_temp_gdt is sized to hold only a null descriptor in slot zero
         * and a 64-bit code descriptor in slot one.
         *
         * rm_temp_[gi]dt_lim and rm_temp_[gi]dt_base are the pseudo-descriptors
         * for the temporary GDT and IDT, respectively.
         */
        uint64_t        rm_temp_gdt[2];
        ushort_t        rm_temp_gdtdesc_pad;    /* filler to align GDT desc */
        ushort_t        rm_temp_gdt_lim;
        uint32_t        rm_temp_gdt_base;
        ushort_t        rm_temp_idtdesc_pad;    /* filler to align IDT desc */
        ushort_t        rm_temp_idt_lim;
        uint32_t        rm_temp_idt_base;

        /*
         * The code executing in the rm_platter needs the offset into the
         * platter at which the 64-bit code starts, so have mp_startup
         * calculate it and store it here.
         */
        uint32_t        rm_longmode64_addr;
#endif  /* __amd64 */
} rm_platter_t;

/*
 * cpu tables put within a single structure two of the tables which need to be
 * allocated when a CPU starts up.
 *
 * Note: the tss should be 16 byte aligned for best performance on amd64
 * Since DEFAULTSTKSIZE is a multiple of PAGESIZE tss will be aligned.
 */
struct cpu_tables {
        /* IST stacks */
        char            ct_stack1[DEFAULTSTKSZ];        /* dblfault */
#if !defined(__xpv)
        char            ct_stack2[DEFAULTSTKSZ];        /* nmi */
        char            ct_stack3[DEFAULTSTKSZ];        /* mce */
#endif
        char            ct_stack[DEFAULTSTKSZ];
        tss_t           ct_tss;
};
______unchanged_portion_omitted_
```

```
**********************************************************
  131966 Fri Apr  6 17:25:04 2018
new/usr/src/uts/i86pc/vm/hat_i86.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9208 hati_demap_func should take pagesize into account
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Tim Kordas <tim.kordas@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */
/*
 * Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
 * Copyright 2018 Joyent, Inc.  All rights reserved.
 * Copyright (c) 2014, 2015 by Delphix. All rights reserved.
 */

/*
 * VM - Hardware Address Translation management for i386 and amd64
 *
 * Implementation of the interfaces described in <common/vm/hat.h>
 *
 * Nearly all the details of how the hardware is managed should not be
 * visible outside this layer except for misc. machine specific functions
 * that work in conjunction with this code.
 *
 * Routines used only inside of i86pc/vm start with hati_ for HAT Internal.
 */

/*
 * amd64 HAT Design
 *
 * ----------
 * Background
 * ----------
 *
 * On x86, the address space is shared between a user process and the kernel.
 * This is different from SPARC. Conventionally, the kernel lives at the top of
 * the address space and the user process gets to enjoy the rest of it. If you
```



```
 * look at the image of the address map in uts/i86pc/os/startup.c, you'll get a
 * rough sense of how the address space is laid out and used.
 *
 * Every unique address space is represented by an instance of a HAT structure
 * called a 'hat_t'. In addition to a hat_t structure for each process, there is
 * also one that is used for the kernel (kas.a_hat), and each CPU ultimately
 * also has a HAT.
 *
 * Each HAT contains a pointer to its root page table. This root page table is
 * what we call an L3 page table in illumos and Intel calls the PML4. It is the
 * physical address of the L3 table that we place in the %cr3 register which the
 * processor uses.
 *
 * Each of the many layers of the page table is represented by a structure
 * called an htable_t. The htable_t manages a set of 512 8-byte entries. The
 * number of entries in a given page table is constant across all different
 * level page tables. Note, this is only true on amd64. This has not always been
 * the case on x86.
 *
 * Each entry in a page table, generally referred to as a PTE, may refer to
 * another page table or a memory location, depending on the level of the page
 * table and the use of large pages. Importantly, the top-level L3 page table
 * (PML4) only supports linking to further page tables. This is also true on
 * systems which support a 5th level page table (which we do not currently
 * support).
 *
 * Historically, on x86, when a process was running on CPU, the root of the page
 * table was inserted into %cr3 on each CPU on which it was currently running.
 * When processes would switch (by calling hat_switch()), then the value in %cr3
 * on that CPU would change to that of the new HAT. While this behavior is still
 * maintained in the xpv kernel, this is not what is done today.
 *
 * ------------------
 * Per-CPU Page Tables
 * ------------------
 *
 * Throughout the system the 64-bit kernel has a notion of what it calls a
 * per-CPU page table or PCP. The notion of a per-CPU page table was originally
 * introduced as part of the original work to support x86 PAE. On the 64-bit
 * kernel, it was originally used for 32-bit processes running on the 64-bit
 * kernel. The rationale behind this was that each 32-bit process could have all
 * of its memory represented in a single L2 page table as each L2 page table
 * entry represents 1 GbE of memory.
 *
 * Following on from this, the idea was that given that all of the L3 page table
 * entries for 32-bit processes are basically going to be identical with the
 * exception of the first entry in the page table, why not share those page
 * table entries. This gave rise to the idea of a per-CPU page table.
 *
 * The way this works is that we have a member in the machcpu_t called the
 * mcpu_hat_info. That structure contains two different 4k pages: one that
 * represents the L3 page table and one that represents an L2 page table. When
 * the CPU starts up, the L3 page table entries are copied in from the kernel's
 * page table. The L3 kernel entries do not change throughout the lifetime of
 * the kernel. The kernel portion of these L3 pages for each CPU have the same
 * records, meaning that they point to the same L2 page tables and thus see a
 * consistent view of the world.
 *
 * When a 32-bit process is loaded into this world, we copy the 32-bit process's
 * four top-level page table entries into the CPU's L2 page table and then set
 * the CPU's first L3 page table entry to point to the CPU's L2 page.
 * Specifically, in hat_pcp_update(), we're copying from the process's
 * HAT_COPIED_32 HAT into the page tables specific to this CPU.
 *
 * As part of the implementation of kernel page table isolation, this was also
 * extended to 64-bit processes. When a 64-bit process runs, we'll copy their L3
```

```
 * PTEs across into the current CPU's L3 page table. (As we can't do the
 * first-L3-entry trick for 64-bit processes, ->hci_pcp_l2ptes is unused in this
 * case.)
 *
 * The use of per-CPU page tables has a lot of implementation ramifications. A
 * HAT that runs a user process will be flagged with the HAT_COPIED flag to
 * indicate that it is using the per-CPU page table functionality. In tandem
 * with the HAT, the top-level htable_t will be flagged with the HTABLE_COPIED
 * flag. If the HAT represents a 32-bit process, then we will also set the
 * HAT_COPIED_32 flag on that hat_t.
 *
 * These two flags work together. The top-level htable_t when using per-CPU page
 * tables is 'virtual'. We never allocate a ptable for this htable_t (i.e.
 * ht->ht_pfn is PFN_INVALID).  Instead, when we need to modify a PTE in an
 * HTABLE_COPIED ptable, x86pte_access_pagetable() will redirect any accesses to
 * ht_hat->hat_copied_ptes.
 *
 * Of course, such a modification won't actually modify the HAT_PCP page tables
 * that were copied from the HAT_COPIED htable. When we change the top level
 * page table entries (L2 PTEs for a 32-bit process and L3 PTEs for a 64-bit
 * process), we need to make sure to trigger hat_pcp_update() on all CPUs that
 * are currently tied to this HAT (including the current CPU).
 *
 * To do this, PCP piggy-backs on TLB invalidation, specifically via the
 * hat_tlb_inval() path from link_ptp() and unlink_ptp().
 *
 * (Importantly, in all such cases, when this is in operation, the top-level
 * entry should not be able to refer to an actual page table entry that can be
 * changed and consolidated into a large page. If large page consolidation is
 * required here, then there will be much that needs to be reconsidered.)
 *
 * ----------------------------------------------
 * Kernel Page Table Isolation and the Per-CPU HAT
 * ----------------------------------------------
 *
 * All Intel CPUs that support speculative execution and paging are subject to a
 * series of bugs that have been termed 'Meltdown'. These exploits allow a user
 * process to read kernel memory through cache side channels and speculative
 * execution. To mitigate this on vulnerable CPUs, we need to use a technique
 * called kernel page table isolation. What this requires is that we have two
 * different page table roots. When executing in kernel mode, we will use a %cr3
 * value that has both the user and kernel pages. However when executing in user
 * mode, we will need to have a %cr3 that has all of the user pages; however,
 * only a subset of the kernel pages required to operate.
 *
 * These kernel pages that we need mapped are:
 *
 *   o Kernel Text that allows us to switch between the cr3 values.
 *   o The current global descriptor table (GDT)
 *   o The current interrupt descriptor table (IDT)
 *   o The current task switching state (TSS)
 *   o The current local descriptor table (LDT)
 *   o Stacks and scratch space used by the interrupt handlers
 *
 * For more information on the stack switching techniques, construction of the
 * trampolines, and more, please see i86pc/ml/kpti_trampolines.s. The most
 * important part of these mappings are the following two constraints:
 *
 *   o The mappings are all per-CPU (except for read-only text)
 *   o The mappings are static. They are all established before the CPU is
 *     started (with the exception of the boot CPU).
 *
 * To facilitate the kernel page table isolation we employ our per-CPU
 * page tables discussed in the previous section and add the notion of a per-CPU
 * HAT. Fundamentally we have a second page table root. There is both a kernel
 * page table (hci_pcp_l3ptes), and a user L3 page table (hci_user_l3ptes).
```



```
 * Both will have the user page table entries copied into them, the same way
 * that we discussed in the section 'Per-CPU Page Tables'.
 *
 * The complex part of this is how do we construct the set of kernel mappings
 * that should be present when running with the user page table. To answer that,
 * we add the notion of a per-CPU HAT. This HAT functions like a normal HAT,
 * except that it's not really associated with an address space the same way
 * that other HATs are.
 *
 * This HAT lives off of the 'struct hat_cpu_info' which is a member of the
 * machcpu in the member hci_user_hat. We use this per-CPU HAT to create the set
 * of kernel mappings that should be present on this CPU. The kernel mappings
 * are added to the per-CPU HAT through the function hati_cpu_punchin(). Once a
 * mapping has been punched in, it may not be punched out. The reason that we
 * opt to leverage a HAT structure is that it knows how to allocate and manage
 * all of the lower level page tables as required.
 *
 * Because all of the mappings are present at the beginning of time for this CPU
 * and none of the mappings are in the kernel pageable segment, we don't have to
 * worry about faulting on these HAT structures and thus the notion of the
 * current HAT that we're using is always the appropriate HAT for the process
 * (usually a user HAT or the kernel's HAT).
 *
 * A further constraint we place on the system with these per-CPU HATs is that
 * they are not subject to htable_steal(). Because each CPU will have a rather
 * fixed number of page tables, the same way that we don't steal from the
 * kernel's HAT, it was determined that we should not steal from this HAT due to
 * the complications involved and somewhat criminal nature of htable_steal().
 *
 * The per-CPU HAT is initialized in hat_pcp_setup() which is called as part of
 * onlining the CPU, but before the CPU is actually started. The per-CPU HAT is
 * removed in hat_pcp_teardown() which is called when a CPU is being offlined to
 * be removed from the system (which is different from what psradm usually
 * does).
 *
 * Finally, once the CPU has been onlined, the set of mappings in the per-CPU
 * HAT must not change. The HAT related functions that we call are not meant to
 * be called when we're switching between processes. For example, it is quite
 * possible that if they were, they would try to grab an htable mutex which
 * another thread might have. One needs to treat hat_switch() as though they
 * were above LOCK_LEVEL and therefore _must not_ block under any circumstance.
 */

#include <sys/machparam.h>
#include <sys/machsystm.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/systm.h>
#include <sys/cpuvar.h>
#include <sys/thread.h>
#include <sys/proc.h>
#include <sys/cpu.h>
#include <sys/kmem.h>
#include <sys/disp.h>
#include <sys/shm.h>
#include <sys/sysmacros.h>
#include <sys/machparam.h>
#include <sys/vmem.h>
#include <sys/vmsystm.h>
#include <sys/promif.h>
#include <sys/var.h>
#include <sys/x86_archext.h>
#include <sys/atomic.h>
#include <sys/bitmap.h>
#include <sys/controlregs.h>
#include <sys/bootconf.h>
```

```
 254 #include <sys/bootsvcs.h>
 255 #include <sys/bootinfo.h>
 256 #include <sys/archsystm.h>

 258 #include <vm/seg_kmem.h>
 259 #include <vm/hat_i86.h>
 260 #include <vm/as.h>
 261 #include <vm/seg.h>
 262 #include <vm/page.h>
 263 #include <vm/seg_kp.h>
 264 #include <vm/seg_kpm.h>
 265 #include <vm/vm_dep.h>
 266 #ifdef __xpv
 267 #include <sys/hypervisor.h>
 268 #endif
 269 #include <vm/kboot_mmu.h>
 270 #include <vm/seg_spt.h>

 272 #include <sys/cmn_err.h>

 274 /*
 275  * Basic parameters for hat operation.
 276  */
 277 struct hat_mmu_info mmu;

 279 /*
 280  * The page that is the kernel's top level pagetable.
 281  *
 282  * For 32 bit PAE support on i86pc, the kernel hat will use the 1st 4 entries
 283  * on this 4K page for its top level page table. The remaining groups of
 284  * 4 entries are used for per processor copies of user PCP pagetables for
  98  * 4 entries are used for per processor copies of user VLP pagetables for
 285  * running threads.  See hat_switch() and reload_pae32() for details.
 286  *
 287  * pcp_page[0..3] - level==2 PTEs for kernel HAT
 288  * pcp_page[4..7] - level==2 PTEs for user thread on cpu 0
 289  * pcp_page[8..11]  - level==2 PTE for user thread on cpu 1
 101  * vlp_page[0..3] - level==2 PTEs for kernel HAT
 102  * vlp_page[4..7] - level==2 PTEs for user thread on cpu 0
 103  * vlp_page[8..11]  - level==2 PTE for user thread on cpu 1
 290  * etc...
 291  *
 292  * On the 64-bit kernel, this is the normal root of the page table and there is
 293  * nothing special about it when used for other CPUs.
 294  */
 295 static x86pte_t *pcp_page;
 106 static x86pte_t *vlp_page;

 297 /*
 298  * forward declaration of internal utility routines
 299  */
 300 static x86pte_t hati_update_pte(htable_t *ht, uint_t entry, x86pte_t expected,
 301         x86pte_t new);

 303 /*
 304  * The kernel address space exists in all non-HAT_COPIED HATs. To implement this
 305  * the kernel reserves a fixed number of entries in the topmost level(s) of page
 306  * tables. The values are setup during startup and then copied to every user hat
 307  * created by hat_alloc(). This means that kernelbase must be:
 115  * The kernel address space exists in all HATs. To implement this the
 116  * kernel reserves a fixed number of entries in the topmost level(s) of page
 117  * tables. The values are setup during startup and then copied to every user
 118  * hat created by hat_alloc(). This means that kernelbase must be:
 308  *
 309  *        4Meg aligned for 32 bit kernels
 310  *      512Gig aligned for x86_64 64 bit kernel
```



```
 311  *
 312  * The hat_kernel_range_ts describe what needs to be copied from kernel hat
 313  * to each user hat.
 314  */
 315 typedef struct hat_kernel_range {
 316         level_t         hkr_level;
 317         uintptr_t       hkr_start_va;
 318         uintptr_t       hkr_end_va;     /* zero means to end of memory */
 319 } hat_kernel_range_t;
 320 #define NUM_KERNEL_RANGE 2
 321 static hat_kernel_range_t kernel_ranges[NUM_KERNEL_RANGE];
 322 static int num_kernel_ranges;

 324 uint_t use_boot_reserve = 1;    /* cleared after early boot process */
 325 uint_t can_steal_post_boot = 0; /* set late in boot to enable stealing */

 327 /*
 328  * enable_1gpg: controls 1g page support for user applications.
 329  * By default, 1g pages are exported to user applications. enable_1gpg can
 330  * be set to 0 to not export.
 331  */
 332 int     enable_1gpg = 1;

 334 /*
 335  * AMD shanghai processors provide better management of 1gb ptes in its tlb.
 336  * By default, 1g page support will be disabled for pre-shanghai AMD
 337  * processors that don't have optimal tlb support for the 1g page size.
 338  * chk_optimal_1gtlb can be set to 0 to force 1g page support on sub-optimal
 339  * processors.
 340  */
 341 int     chk_optimal_1gtlb = 1;


 344 #ifdef DEBUG
 345 uint_t  map1gcnt;
 346 #endif


 349 /*
 350  * A cpuset for all cpus. This is used for kernel address cross calls, since
 351  * the kernel addresses apply to all cpus.
 352  */
 353 cpuset_t khat_cpuset;

 355 /*
 356  * management stuff for hat structures
 357  */
 358 kmutex_t        hat_list_lock;
 359 kcondvar_t      hat_list_cv;
 360 kmem_cache_t    *hat_cache;
 361 kmem_cache_t    *hat_hash_cache;
 362 kmem_cache_t    *hat32_hash_cache;
 173 kmem_cache_t    *vlp_hash_cache;

 364 /*
 365  * Simple statistics
 366  */
 367 struct hatstats hatstat;

 369 /*
 370  * Some earlier hypervisor versions do not emulate cmpxchg of PTEs
 371  * correctly.  For such hypervisors we must set PT_USER for kernel
 372  * entries ourselves (normally the emulation would set PT_USER for
 373  * kernel entries and PT_USER|PT_GLOBAL for user entries).  pt_kern is
 374  * thus set appropriately.  Note that dboot/kbm is OK, as only the full
 375  * HAT uses cmpxchg() and the other paths (hypercall etc.) were never
```

```
 * incorrect.
 */
int pt_kern;

/*
 * useful stuff for atomic access/clearing/setting REF/MOD/RO bits in page_t's.
 */
extern void atomic_orb(uchar_t *addr, uchar_t val);
extern void atomic_andb(uchar_t *addr, uchar_t val);

#ifndef __xpv
extern pfn_t memseg_get_start(struct memseg *);
#endif

#define PP_GETRM(pp, rmmask)    (pp->p_nrm & rmmask)
#define PP_ISMOD(pp)            PP_GETRM(pp, P_MOD)
#define PP_ISREF(pp)            PP_GETRM(pp, P_REF)
#define PP_ISRO(pp)             PP_GETRM(pp, P_RO)

#define PP_SETRM(pp, rm)        atomic_orb(&(pp->p_nrm), rm)
#define PP_SETMOD(pp)           PP_SETRM(pp, P_MOD)
#define PP_SETREF(pp)           PP_SETRM(pp, P_REF)
#define PP_SETRO(pp)            PP_SETRM(pp, P_RO)

#define PP_CLRRM(pp, rm)        atomic_andb(&(pp->p_nrm), ~(rm))
#define PP_CLRMOD(pp)           PP_CLRRM(pp, P_MOD)
#define PP_CLRREF(pp)           PP_CLRRM(pp, P_REF)
#define PP_CLRRO(pp)            PP_CLRRM(pp, P_RO)
#define PP_CLRALL(pp)           PP_CLRRM(pp, P_MOD | P_REF | P_RO)

/*
 * kmem cache constructor for struct hat
 */
/*ARGSUSED*/
static int
hati_constructor(void *buf, void *handle, int kmflags)
{
        hat_t   *hat = buf;

        mutex_init(&hat->hat_mutex, NULL, MUTEX_DEFAULT, NULL);
        bzero(hat->hat_pages_mapped,
            sizeof (pgcnt_t) * (mmu.max_page_level + 1));
        hat->hat_ism_pgcnt = 0;
        hat->hat_stats = 0;
        hat->hat_flags = 0;
        CPUSET_ZERO(hat->hat_cpus);
        hat->hat_htable = NULL;
        hat->hat_ht_hash = NULL;
        return (0);
}

/*
 * Put it at the start of the global list of all hats (used by stealing)
 *
 * kas.a_hat is not in the list but is instead used to find the
 * first and last items in the list.
 *
 * - kas.a_hat->hat_next points to the start of the user hats.
 *   The list ends where hat->hat_next == NULL
 *
 * - kas.a_hat->hat_prev points to the last of the user hats.
 *   The list begins where hat->hat_prev == NULL
 */
static void
hat_list_append(hat_t *hat)
{
```



```
        mutex_enter(&hat_list_lock);
        hat->hat_prev = NULL;
        hat->hat_next = kas.a_hat->hat_next;
        if (hat->hat_next)
                hat->hat_next->hat_prev = hat;
        else
                kas.a_hat->hat_prev = hat;
        kas.a_hat->hat_next = hat;
        mutex_exit(&hat_list_lock);
}

/*
 * Allocate a hat structure for as. We also create the top level
 * htable and initialize it to contain the kernel hat entries.
 */
hat_t *
hat_alloc(struct as *as)
{
        hat_t                   *hat;
        htable_t                *ht;    /* top level htable */
        uint_t                  use_copied;
        uint_t                  use_vlp;
        uint_t                  r;
        hat_kernel_range_t      *rp;
        uintptr_t               va;
        uintptr_t               eva;
        uint_t                  start;
        uint_t                  cnt;
        htable_t                *src;
        boolean_t               use_hat32_cache;

        /*
         * Once we start creating user process HATs we can enable
         * the htable_steal() code.
         */
        if (can_steal_post_boot == 0)
                can_steal_post_boot = 1;

        ASSERT(AS_WRITE_HELD(as));
        hat = kmem_cache_alloc(hat_cache, KM_SLEEP);
        hat->hat_as = as;
        mutex_init(&hat->hat_mutex, NULL, MUTEX_DEFAULT, NULL);
        ASSERT(hat->hat_flags == 0);

#if defined(__xpv)
        /*
         * No PCP stuff on the hypervisor due to the 64-bit split top level
         * No VLP stuff on the hypervisor due to the 64-bit split top level
         * page tables.  On 32-bit it's not needed as the hypervisor takes
         * care of copying the top level PTEs to a below 4Gig page.
         */
        use_copied = 0;
        use_hat32_cache = B_FALSE;
        hat->hat_max_level = mmu.max_level;
        hat->hat_num_copied = 0;
        hat->hat_flags = 0;
        use_vlp = 0;
#else   /* __xpv */

        /*
         * All processes use HAT_COPIED on the 64-bit kernel if KPTI is
         * turned on.
         */
        if (ttoproc(curthread)->p_model == DATAMODEL_ILP32) {
                use_copied = 1;
                hat->hat_max_level = mmu.max_level32;
```

```
499                 hat->hat_num_copied = mmu.num_copied_ents32;
500                 use_hat32_cache = B_TRUE;
501                 hat->hat_flags |= HAT_COPIED_32;
502                 HATSTAT_INC(hs_hat_copied32);
503         } else if (kpti_enable == 1) {
504                 use_copied = 1;
505                 hat->hat_max_level = mmu.max_level;
506                 hat->hat_num_copied = mmu.num_copied_ents;
507                 use_hat32_cache = B_FALSE;
508                 HATSTAT_INC(hs_hat_copied64);
509         } else {
510                 use_copied = 0;
511                 use_hat32_cache = B_FALSE;
512                 hat->hat_max_level = mmu.max_level;
513                 hat->hat_num_copied = 0;
514                 hat->hat_flags = 0;
515                 HATSTAT_INC(hs_hat_normal64);
516         }
277         /* 32 bit processes uses a VLP style hat when running with PAE */
278 #if defined(__amd64)
279         use_vlp = (ttoproc(curthread)->p_model == DATAMODEL_ILP32);
280 #elif defined(__i386)
281         use_vlp = mmu.pae_hat;
282 #endif
517 #endif  /* __xpv */
518         if (use_copied) {
519                 hat->hat_flags |= HAT_COPIED;
520                 bzero(hat->hat_copied_ptes, sizeof (hat->hat_copied_ptes));
284         if (use_vlp) {
285                 hat->hat_flags = HAT_VLP;
286                 bzero(hat->hat_vlp_ptes, VLP_SIZE);
521         }

523         /*
524          * Allocate the htable hash. For 32-bit PCP processes we use the
525          * hat32_hash_cache. However, for 64-bit PCP processes we do not as the
526          * number of entries that they have to handle is closer to
527          * hat_hash_cache in count (though there will be more wastage when we
528          * have more DRAM in the system and thus push down the user address
529          * range).
290          * Allocate the htable hash
530          */
531         if (use_hat32_cache) {
532                 hat->hat_num_hash = mmu.hat32_hash_cnt;
533                 hat->hat_ht_hash = kmem_cache_alloc(hat32_hash_cache, KM_SLEEP);
292         if ((hat->hat_flags & HAT_VLP)) {
293                 hat->hat_num_hash = mmu.vlp_hash_cnt;
294                 hat->hat_ht_hash = kmem_cache_alloc(vlp_hash_cache, KM_SLEEP);
534         } else {
535                 hat->hat_num_hash = mmu.hash_cnt;
536                 hat->hat_ht_hash = kmem_cache_alloc(hat_hash_cache, KM_SLEEP);
537         }
538         bzero(hat->hat_ht_hash, hat->hat_num_hash * sizeof (htable_t *));

540         /*
541          * Initialize Kernel HAT entries at the top of the top level page
542          * tables for the new hat.
543          */
544         hat->hat_htable = NULL;
545         hat->hat_ht_cached = NULL;
546         XPV_DISALLOW_MIGRATE();
547         ht = htable_create(hat, (uintptr_t)0, TOP_LEVEL(hat), NULL);
548         hat->hat_htable = ht;

550 #if defined(__amd64)
551         if (hat->hat_flags & HAT_COPIED)
```



```
312         if (hat->hat_flags & HAT_VLP)
552                 goto init_done;
553 #endif

555         for (r = 0; r < num_kernel_ranges; ++r) {
556                 rp = &kernel_ranges[r];
557                 for (va = rp->hkr_start_va; va != rp->hkr_end_va;
558                     va += cnt * LEVEL_SIZE(rp->hkr_level)) {

560                         if (rp->hkr_level == TOP_LEVEL(hat))
561                                 ht = hat->hat_htable;
562                         else
563                                 ht = htable_create(hat, va, rp->hkr_level,
564                                     NULL);

566                         start = htable_va2entry(va, ht);
567                         cnt = HTABLE_NUM_PTES(ht) - start;
568                         eva = va +
569                             ((uintptr_t)cnt << LEVEL_SHIFT(rp->hkr_level));
570                         if (rp->hkr_end_va != 0 &&
571                             (eva > rp->hkr_end_va || eva == 0))
572                                 cnt = htable_va2entry(rp->hkr_end_va, ht) -
573                                     start;

575 #if defined(__i386) && !defined(__xpv)
576                         if (ht->ht_flags & HTABLE_COPIED) {
577                                 bcopy(&pcp_page[start],
578                                     &hat->hat_copied_ptes[start],
337                         if (ht->ht_flags & HTABLE_VLP) {
338                                 bcopy(&vlp_page[start],
339                                     &hat->hat_vlp_ptes[start],
579                                     cnt * sizeof (x86pte_t));
580                                 continue;
581                         }
582 #endif
583                         src = htable_lookup(kas.a_hat, va, rp->hkr_level);
584                         ASSERT(src != NULL);
585                         x86pte_copy(src, ht, start, cnt);
586                         htable_release(src);
587                 }
588         }

590 init_done:

592 #if defined(__xpv)
593         /*
594          * Pin top level page tables after initializing them
595          */
596         xen_pin(hat->hat_htable->ht_pfn, mmu.max_level);
597 #if defined(__amd64)
598         xen_pin(hat->hat_user_ptable, mmu.max_level);
599 #endif
600 #endif
601         XPV_ALLOW_MIGRATE();

603         hat_list_append(hat);

605         return (hat);
606 }

608 #if !defined(__xpv)
609 /*
610  * Cons up a HAT for a CPU. This represents the user mappings. This will have
611  * various kernel pages punched into it manually. Importantly, this hat is
612  * ineligible for stealing. We really don't want to deal with this ever
613  * faulting and figuring out that this is happening, much like we don't with
```

```
614  * kas.
615  */
616 static hat_t *
617 hat_cpu_alloc(cpu_t *cpu)
618 {
619         hat_t *hat;
620         htable_t *ht;

622         hat = kmem_cache_alloc(hat_cache, KM_SLEEP);
623         hat->hat_as = NULL;
624         mutex_init(&hat->hat_mutex, NULL, MUTEX_DEFAULT, NULL);
625         hat->hat_max_level = mmu.max_level;
626         hat->hat_num_copied = 0;
627         hat->hat_flags = HAT_PCP;

629         hat->hat_num_hash = mmu.hash_cnt;
630         hat->hat_ht_hash = kmem_cache_alloc(hat_hash_cache, KM_SLEEP);
631         bzero(hat->hat_ht_hash, hat->hat_num_hash * sizeof (htable_t *));

633         hat->hat_next = hat->hat_prev = NULL;

635         /*
636          * Because this HAT will only ever be used by the current CPU, we'll go
637          * ahead and set the CPUSET up to only point to the CPU in question.
365          * Put it at the start of the global list of all hats (used by stealing)
366          *
367          * kas.a_hat is not in the list but is instead used to find the
368          * first and last items in the list.
369          *
370          * - kas.a_hat->hat_next points to the start of the user hats.
371          *   The list ends where hat->hat_next == NULL
372          *
373          * - kas.a_hat->hat_prev points to the last of the user hats.
374          *   The list begins where hat->hat_prev == NULL
638          */
639         CPUSET_ADD(hat->hat_cpus, cpu->cpu_id);
376         mutex_enter(&hat_list_lock);
377         hat->hat_prev = NULL;
378         hat->hat_next = kas.a_hat->hat_next;
379         if (hat->hat_next)
380                 hat->hat_next->hat_prev = hat;
381         else
382                 kas.a_hat->hat_prev = hat;
383         kas.a_hat->hat_next = hat;
384         mutex_exit(&hat_list_lock);

641         hat->hat_htable = NULL;
642         hat->hat_ht_cached = NULL;
643         ht = htable_create(hat, (uintptr_t)0, TOP_LEVEL(hat), NULL);
644         hat->hat_htable = ht;

646         hat_list_append(hat);

648         return (hat);
649 }
650 #endif /* !__xpv */

652 /*
653  * process has finished executing but as has not been cleaned up yet.
654  */
655 /*ARGSUSED*/
656 void
657 hat_free_start(hat_t *hat)
658 {
659         ASSERT(AS_WRITE_HELD(hat->hat_as));
```



```
661         /*
662          * If the hat is currently a stealing victim, wait for the stealing
663          * to finish.  Once we mark it as HAT_FREEING, htable_steal()
664          * won't look at its pagetables anymore.
665          */
666         mutex_enter(&hat_list_lock);
667         while (hat->hat_flags & HAT_VICTIM)
668                 cv_wait(&hat_list_cv, &hat_list_lock);
669         hat->hat_flags |= HAT_FREEING;
670         mutex_exit(&hat_list_lock);
671 }

673 /*
674  * An address space is being destroyed, so we destroy the associated hat.
675  */
676 void
677 hat_free_end(hat_t *hat)
678 {
679         kmem_cache_t *cache;

681         ASSERT(hat->hat_flags & HAT_FREEING);

683         /*
684          * must not be running on the given hat
685          */
686         ASSERT(CPU->cpu_current_hat != hat);

688         /*
689          * Remove it from the list of HATs
690          */
691         mutex_enter(&hat_list_lock);
692         if (hat->hat_prev)
693                 hat->hat_prev->hat_next = hat->hat_next;
694         else
695                 kas.a_hat->hat_next = hat->hat_next;
696         if (hat->hat_next)
697                 hat->hat_next->hat_prev = hat->hat_prev;
698         else
699                 kas.a_hat->hat_prev = hat->hat_prev;
700         mutex_exit(&hat_list_lock);
701         hat->hat_next = hat->hat_prev = NULL;

703 #if defined(__xpv)
704         /*
705          * On the hypervisor, unpin top level page table(s)
706          */
707         VERIFY3U(hat->hat_flags & HAT_PCP, ==, 0);
708         xen_unpin(hat->hat_htable->ht_pfn);
709 #if defined(__amd64)
710         xen_unpin(hat->hat_user_ptable);
711 #endif
712 #endif

714         /*
715          * Make a pass through the htables freeing them all up.
716          */
717         htable_purge_hat(hat);

719         /*
720          * Decide which kmem cache the hash table came from, then free it.
721          */
722         if (hat->hat_flags & HAT_COPIED) {
723 #if defined(__amd64)
724                 if (hat->hat_flags & HAT_COPIED_32) {
725                         cache = hat32_hash_cache;
726                 } else {
```

```
458         if (hat->hat_flags & HAT_VLP)
459                 cache = vlp_hash_cache;
460         else
727                         cache = hat_hash_cache;
728                 }
729 #else
730                 cache = hat32_hash_cache;
731 #endif
732         } else {
733                 cache = hat_hash_cache;
734         }
735         kmem_cache_free(cache, hat->hat_ht_hash);
736         hat->hat_ht_hash = NULL;

738         hat->hat_flags = 0;
739         hat->hat_max_level = 0;
740         hat->hat_num_copied = 0;
741         kmem_cache_free(hat_cache, hat);
742 }
_____unchanged_portion_omitted_

794 /*
795  * Determine the number of slots that are in used in the top-most level page
796  * table for user memory. This is based on _userlimit. In effect this is similar
797  * to htable_va2entry, but without the convenience of having an htable.
798  */
799 void
800 mmu_calc_user_slots(void)
801 {
802         uint_t ent, nptes;
803         uintptr_t shift;

805         nptes = mmu.top_level_count;
806         shift = _userlimit >> mmu.level_shift[mmu.max_level];
807         ent = shift & (nptes - 1);

809         /*
810          * Ent tells us the slot that the page for _userlimit would fit in. We
811          * need to add one to this to cover the total number of entries.
812          */
813         mmu.top_level_uslots = ent + 1;

815         /*
816          * When running 32-bit compatability processes on a 64-bit kernel, we
817          * will only need to use one slot.
818          */
819         mmu.top_level_uslots32 = 1;

821         /*
822          * Record the number of PCP page table entries that we'll need to copy
823          * around. For 64-bit processes this is the number of user slots. For
824          * 32-bit proceses, this is 4 1 GiB pages.
825          */
826         mmu.num_copied_ents = mmu.top_level_uslots;
827         mmu.num_copied_ents32 = 4;
828 }

830 /*
831  * Initialize hat data structures based on processor MMU information.
832  */
833 void
834 mmu_init(void)
835 {
836         uint_t max_htables;
837         uint_t pa_bits;
838         uint_t va_bits;
```



```
839         int i;

841         /*
842          * If CPU enabled the page table global bit, use it for the kernel
843          * This is bit 7 in CR4 (PGE - Page Global Enable).
844          */
845         if (is_x86_feature(x86_featureset, X86FSET_PGE) &&
846             (getcr4() & CR4_PGE) != 0)
847                 mmu.pt_global = PT_GLOBAL;

849 #if !defined(__xpv)
850         /*
851          * The 64-bit x86 kernel has split user/kernel page tables. As such we
852          * cannot have the global bit set. The simplest way for us to deal with
853          * this is to just say that pt_global is zero, so the global bit isn't
854          * present.
855          */
856         if (kpti_enable == 1)
857                 mmu.pt_global = 0;
858 #endif

860         /*
861          * Detect NX and PAE usage.
862          */
863         mmu.pae_hat = kbm_pae_support;
864         if (kbm_nx_support)
865                 mmu.pt_nx = PT_NX;
866         else
867                 mmu.pt_nx = 0;

869         /*
870          * Use CPU info to set various MMU parameters
871          */
872         cpuid_get_addrsize(CPU, &pa_bits, &va_bits);

874         if (va_bits < sizeof (void *) * NBBY) {
875                 mmu.hole_start = (1ul << (va_bits - 1));
876                 mmu.hole_end = 0ul - mmu.hole_start - 1;
877         } else {
878                 mmu.hole_end = 0;
879                 mmu.hole_start = mmu.hole_end - 1;
880         }
881 #if defined(OPTERON_ERRATUM_121)
882         /*
883          * If erratum 121 has already been detected at this time, hole_start
884          * contains the value to be subtracted from mmu.hole_start.
885          */
886         ASSERT(hole_start == 0 || opteron_erratum_121 != 0);
887         hole_start = mmu.hole_start - hole_start;
888 #else
889         hole_start = mmu.hole_start;
890 #endif
891         hole_end = mmu.hole_end;

893         mmu.highest_pfn = mmu_btop((1ull << pa_bits) - 1);
894         if (mmu.pae_hat == 0 && pa_bits > 32)
895                 mmu.highest_pfn = PFN_4G - 1;

897         if (mmu.pae_hat) {
898                 mmu.pte_size = 8;       /* 8 byte PTEs */
899                 mmu.pte_size_shift = 3;
900         } else {
901                 mmu.pte_size = 4;       /* 4 byte PTEs */
902                 mmu.pte_size_shift = 2;
903         }
```

```
905         if (mmu.pae_hat && !is_x86_feature(x86_featureset, X86FSET_PAE))
906                 panic("Processor does not support PAE");
907
908         if (!is_x86_feature(x86_featureset, X86FSET_CX8))
909                 panic("Processor does not support cmpxchg8b instruction");
910
911 #if defined(__amd64)
912
913         mmu.num_level = 4;
914         mmu.max_level = 3;
915         mmu.ptes_per_table = 512;
916         mmu.top_level_count = 512;
917
918         /*
919          * 32-bit processes only use 1 GB ptes.
920          */
921         mmu.max_level32 = 2;
922
923         mmu.level_shift[0] = 12;
924         mmu.level_shift[1] = 21;
925         mmu.level_shift[2] = 30;
926         mmu.level_shift[3] = 39;
927
928 #elif defined(__i386)
929
930         if (mmu.pae_hat) {
931                 mmu.num_level = 3;
932                 mmu.max_level = 2;
933                 mmu.ptes_per_table = 512;
934                 mmu.top_level_count = 4;
935
936                 mmu.level_shift[0] = 12;
937                 mmu.level_shift[1] = 21;
938                 mmu.level_shift[2] = 30;
939
940         } else {
941                 mmu.num_level = 2;
942                 mmu.max_level = 1;
943                 mmu.ptes_per_table = 1024;
944                 mmu.top_level_count = 1024;
945
946                 mmu.level_shift[0] = 12;
947                 mmu.level_shift[1] = 22;
948         }
949
950 #endif  /* __i386 */
951
952         for (i = 0; i < mmu.num_level; ++i) {
953                 mmu.level_size[i] = 1UL << mmu.level_shift[i];
954                 mmu.level_offset[i] = mmu.level_size[i] - 1;
955                 mmu.level_mask[i] = ~mmu.level_offset[i];
956         }
957
958         set_max_page_level();
959         mmu_calc_user_slots();
960
961         mmu_page_sizes = mmu.max_page_level + 1;
962         mmu_exported_page_sizes = mmu.umax_page_level + 1;
963
964         /* restrict legacy applications from using pagesizes 1g and above */
965         mmu_legacy_page_sizes =
966             (mmu_exported_page_sizes > 2) ? 2 : mmu_exported_page_sizes;
967
968
969         for (i = 0; i <= mmu.max_page_level; ++i) {
970                 mmu.pte_bits[i] = PT_VALID | pt_kern;
```



```
971                 if (i > 0)
972                         mmu.pte_bits[i] |= PT_PAGESIZE;
973         }
974
975         /*
976          * NOTE Legacy 32 bit PAE mode only has the P_VALID bit at top level.
977          */
978         for (i = 1; i < mmu.num_level; ++i)
979                 mmu.ptp_bits[i] = PT_PTPBITS;
980
981 #if defined(__i386)
982         mmu.ptp_bits[2] = PT_VALID;
983 #endif
984
985         /*
986          * Compute how many hash table entries to have per process for htables.
987          * We start with 1 page's worth of entries.
988          *
989          * If physical memory is small, reduce the amount need to cover it.
990          */
991         max_htables = physmax / mmu.ptes_per_table;
992         mmu.hash_cnt = MMU_PAGESIZE / sizeof (htable_t *);
993         while (mmu.hash_cnt > 16 && mmu.hash_cnt >= max_htables)
994                 mmu.hash_cnt >>= 1;
995         mmu.hat32_hash_cnt = mmu.hash_cnt;
667         mmu.vlp_hash_cnt = mmu.hash_cnt;
996
997 #if defined(__amd64)
998         /*
999          * If running in 64 bits and physical memory is large,
1000         * increase the size of the cache to cover all of memory for
1001         * a 64 bit process.
1002         */
1003 #define HASH_MAX_LENGTH 4
1004         while (mmu.hash_cnt * HASH_MAX_LENGTH < max_htables)
1005                 mmu.hash_cnt <<= 1;
1006 #endif
1007 }
1008
1009
1010 /*
1011  * initialize hat data structures
1012  */
1013 void
1014 hat_init()
1015 {
1016 #if defined(__i386)
1017         /*
1018          * _userlimit must be aligned correctly
1019          */
1020         if ((_userlimit & LEVEL_MASK(1)) != _userlimit) {
1021                 prom_printf("hat_init(): _userlimit=%p, not aligned at %p\n",
1022                     (void *)_userlimit, (void *)LEVEL_SIZE(1));
1023                 halt("hat_init(): Unable to continue");
1024         }
1025 #endif
1026
1027         cv_init(&hat_list_cv, NULL, CV_DEFAULT, NULL);
1028
1029         /*
1030          * initialize kmem caches
1031          */
1032         htable_init();
1033         hment_init();
1034
1035         hat_cache = kmem_cache_create("hat_t",
```

```
1036             sizeof (hat_t), 0, hati_constructor, NULL, NULL,
1037             NULL, 0, 0);
1039         hat_hash_cache = kmem_cache_create("HatHash",
1040             mmu.hash_cnt * sizeof (htable_t *), 0, NULL, NULL, NULL,
1041             NULL, 0, 0);
1043         /*
1044          * 32-bit PCP hats can use a smaller hash table size on large memory
1045          * machines
 716          * VLP hats can use a smaller hash table size on large memroy machines
1046          */
1047         if (mmu.hash_cnt == mmu.hat32_hash_cnt) {
1048                 hat32_hash_cache = hat_hash_cache;
 718         if (mmu.hash_cnt == mmu.vlp_hash_cnt) {
 719                 vlp_hash_cache = hat_hash_cache;
1049         } else {
1050                 hat32_hash_cache = kmem_cache_create("Hat32Hash",
1051                     mmu.hat32_hash_cnt * sizeof (htable_t *), 0, NULL, NULL,
1052                     NULL, NULL, 0, 0);
 721                 vlp_hash_cache = kmem_cache_create("HatVlpHash",
 722                     mmu.vlp_hash_cnt * sizeof (htable_t *), 0, NULL, NULL, NULL,
 723                     NULL, 0, 0);
1053         }
1055         /*
1056          * Set up the kernel's hat
1057          */
1058         AS_LOCK_ENTER(&kas, RW_WRITER);
1059         kas.a_hat = kmem_cache_alloc(hat_cache, KM_NOSLEEP);
1060         mutex_init(&kas.a_hat->hat_mutex, NULL, MUTEX_DEFAULT, NULL);
1061         kas.a_hat->hat_as = &kas;
1062         kas.a_hat->hat_flags = 0;
1063         AS_LOCK_EXIT(&kas);
1065         CPUSET_ZERO(khat_cpuset);
1066         CPUSET_ADD(khat_cpuset, CPU->cpu_id);
1068         /*
1069          * The kernel HAT doesn't use PCP regardless of architectures.
1070          */
1071         ASSERT3U(mmu.max_level, >, 0);
1072         kas.a_hat->hat_max_level = mmu.max_level;
1073         kas.a_hat->hat_num_copied = 0;
1075         /*
1076          * The kernel hat's next pointer serves as the head of the hat list .
1077          * The kernel hat's prev pointer tracks the last hat on the list for
1078          * htable_steal() to use.
1079          */
1080         kas.a_hat->hat_next = NULL;
1081         kas.a_hat->hat_prev = NULL;
1083         /*
1084          * Allocate an htable hash bucket for the kernel
1085          * XX64 - tune for 64 bit procs
1086          */
1087         kas.a_hat->hat_num_hash = mmu.hash_cnt;
1088         kas.a_hat->hat_ht_hash = kmem_cache_alloc(hat_hash_cache, KM_NOSLEEP);
1089         bzero(kas.a_hat->hat_ht_hash, mmu.hash_cnt * sizeof (htable_t *));
1091         /*
1092          * zero out the top level and cached htable pointers
1093          */
1094         kas.a_hat->hat_ht_cached = NULL;
1095         kas.a_hat->hat_htable = NULL;
```



```
1097         /*
1098          * Pre-allocate hrm_hashtab before enabling the collection of
1099          * refmod statistics.  Allocating on the fly would mean us
1100          * running the risk of suffering recursive mutex enters or
1101          * deadlocks.
1102          */
1103         hrm_hashtab = kmem_zalloc(HRM_HASHSIZE * sizeof (struct hrmstat *),
1104             KM_SLEEP);
1105 }

1108 extern void kpti_tramp_start();
1109 extern void kpti_tramp_end();

1111 extern void kdi_isr_start();
1112 extern void kdi_isr_end();

1114 extern gate_desc_t kdi_idt[NIDT];

1116 /*
1117  * Prepare per-CPU pagetables for all processes on the 64 bit kernel.
 772  * Prepare CPU specific pagetables for VLP processes on 64 bit kernels.
1118  *
1119  * Each CPU has a set of 2 pagetables that are reused for any 32 bit
1120  * process it runs. They are the top level pagetable, hci_pcp_l3ptes, and
1121  * the next to top level table for the bottom 512 Gig, hci_pcp_l2ptes.
 775  * process it runs. They are the top level pagetable, hci_vlp_l3ptes, and
 776  * the next to top level table for the bottom 512 Gig, hci_vlp_l2ptes.
1122  */
1123 /*ARGSUSED*/
1124 static void
1125 hat_pcp_setup(struct cpu *cpu)
 780 hat_vlp_setup(struct cpu *cpu)
1126 {
1127 #if !defined(__xpv)
 782 #if defined(__amd64) && !defined(__xpv)
1128         struct hat_cpu_info *hci = cpu->cpu_hat_info;
1129         uintptr_t va;
1130         size_t len;
 784         pfn_t pfn;
1132         /*
1133          * allocate the level==2 page table for the bottom most
1134          * 512Gig of address space (this is where 32 bit apps live)
1135          */
1136         ASSERT(hci != NULL);
1137         hci->hci_pcp_l2ptes = kmem_zalloc(MMU_PAGESIZE, KM_SLEEP);
 791         hci->hci_vlp_l2ptes = kmem_zalloc(MMU_PAGESIZE, KM_SLEEP);
1139         /*
1140          * Allocate a top level pagetable and copy the kernel's
1141          * entries into it. Then link in hci_pcp_l2ptes in the 1st entry.
 795          * entries into it. Then link in hci_vlp_l2ptes in the 1st entry.
1142          */
1143         hci->hci_pcp_l3ptes = kmem_zalloc(MMU_PAGESIZE, KM_SLEEP);
1144         hci->hci_pcp_l3pfn =
1145             hat_getpfnum(kas.a_hat, (caddr_t)hci->hci_pcp_l3ptes);
1146         ASSERT3U(hci->hci_pcp_l3pfn, !=, PFN_INVALID);
1147         bcopy(pcp_page, hci->hci_pcp_l3ptes, MMU_PAGESIZE);
 797         hci->hci_vlp_l3ptes = kmem_zalloc(MMU_PAGESIZE, KM_SLEEP);
 798         hci->hci_vlp_pfn =
 799             hat_getpfnum(kas.a_hat, (caddr_t)hci->hci_vlp_l3ptes);
 800         ASSERT(hci->hci_vlp_pfn != PFN_INVALID);
 801         bcopy(vlp_page, hci->hci_vlp_l3ptes, MMU_PAGESIZE);
```

```
1149         hci->hci_pcp_l2pfn =
1150             hat_getpfnum(kas.a_hat, (caddr_t)hci->hci_pcp_l2ptes);
1151         ASSERT3U(hci->hci_pcp_l2pfn, !=, PFN_INVALID);
1152
1153         /*
1154          * Now go through and allocate the user version of these structures.
1155          * Unlike with the kernel version, we allocate a hat to represent the
1156          * top-level page table as that will make it much simpler when we need
1157          * to patch through user entries.
1158          */
1159         hci->hci_user_hat = hat_cpu_alloc(cpu);
1160         hci->hci_user_l3pfn = hci->hci_user_hat->hat_htable->ht_pfn;
1161         ASSERT3U(hci->hci_user_l3pfn, !=, PFN_INVALID);
1162         hci->hci_user_l3ptes =
1163             (x86pte_t *)hat_kpm_mapin_pfn(hci->hci_user_l3pfn);
1164
1165         /* Skip the rest of this if KPTI is switched off at boot. */
1166         if (kpti_enable != 1)
1167                 return;
1168
1169         /*
1170          * OK, now that we have this we need to go through and punch the normal
1171          * holes in the CPU's hat for this. At this point we'll punch in the
1172          * following:
1173          *
1174          *   o GDT
1175          *   o IDT
1176          *   o LDT
1177          *   o Trampoline Code
1178          *   o machcpu KPTI page
1179          *   o kmdb ISR code page (just trampolines)
1180          *
1181          * If this is cpu0, then we also can initialize the following because
1182          * they'll have already been allocated.
1183          *
1184          *   o TSS for CPU 0
1185          *   o Double Fault for CPU 0
1186          *
1187          * The following items have yet to be allocated and have not been
1188          * punched in yet. They will be punched in later:
1189          *
1190          *   o TSS (mach_cpucontext_alloc_tables())
1191          *   o Double Fault Stack (mach_cpucontext_alloc_tables())
1192          */
1193         hati_cpu_punchin(cpu, (uintptr_t)cpu->cpu_gdt, PROT_READ);
1194         hati_cpu_punchin(cpu, (uintptr_t)cpu->cpu_idt, PROT_READ);
1195
1196         /*
1197          * As the KDI IDT is only active during kmdb sessions (including single
1198          * stepping), typically we don't actually need this punched in (we
1199          * consider the routines that switch to the user cr3 to be toxic).  But
1200          * if we ever accidentally end up on the user cr3 while on this IDT,
1201          * we'd prefer not to triple fault.
1202          */
1203         hati_cpu_punchin(cpu, (uintptr_t)&kdi_idt, PROT_READ);
1204
1205         CTASSERT(((uintptr_t)&kpti_tramp_start % MMU_PAGESIZE) == 0);
1206         CTASSERT(((uintptr_t)&kpti_tramp_end % MMU_PAGESIZE) == 0);
1207         for (va = (uintptr_t)&kpti_tramp_start;
1208             va < (uintptr_t)&kpti_tramp_end; va += MMU_PAGESIZE) {
1209                 hati_cpu_punchin(cpu, va, PROT_READ | PROT_EXEC);
1210         }
1211
1212         VERIFY3U(((uintptr_t)cpu->cpu_m.mcpu_ldt) % MMU_PAGESIZE, ==, 0);
1213         for (va = (uintptr_t)cpu->cpu_m.mcpu_ldt, len = LDT_CPU_SIZE;
1214             len >= MMU_PAGESIZE; va += MMU_PAGESIZE, len -= MMU_PAGESIZE) {
```



```
1215                 hati_cpu_punchin(cpu, va, PROT_READ);
1216         }
1217
1218         /* mcpu_pad2 is the start of the page containing the kpti_frames. */
1219         hati_cpu_punchin(cpu, (uintptr_t)&cpu->cpu_m.mcpu_pad2[0],
1220             PROT_READ | PROT_WRITE);
1221
1222         if (cpu == &cpus[0]) {
1223                 /*
1224                  * CPU0 uses a global for its double fault stack to deal with
1225                  * the chicken and egg problem. We need to punch it into its
1226                  * user HAT.
1227                  */
1228                 extern char dblfault_stack0[];
1229
1230                 hati_cpu_punchin(cpu, (uintptr_t)cpu->cpu_m.mcpu_tss,
1231                     PROT_READ);
1232
1233                 for (va = (uintptr_t)dblfault_stack0,
1234                     len = DEFAULTSTKSZ; len >= MMU_PAGESIZE;
1235                     va += MMU_PAGESIZE, len -= MMU_PAGESIZE) {
1236                         hati_cpu_punchin(cpu, va, PROT_READ | PROT_WRITE);
1237                 }
1238         }
1239
1240         CTASSERT(((uintptr_t)&kdi_isr_start % MMU_PAGESIZE) == 0);
1241         CTASSERT(((uintptr_t)&kdi_isr_end % MMU_PAGESIZE) == 0);
1242         for (va = (uintptr_t)&kdi_isr_start;
1243             va < (uintptr_t)&kdi_isr_end; va += MMU_PAGESIZE) {
1244                 hati_cpu_punchin(cpu, va, PROT_READ | PROT_EXEC);
1245         }
1246 #endif /* !__xpv */
 803         pfn = hat_getpfnum(kas.a_hat, (caddr_t)hci->hci_vlp_l2ptes);
 804         ASSERT(pfn != PFN_INVALID);
 805         hci->hci_vlp_l3ptes[0] = MAKEPTP(pfn, 2);
 806 #endif /* __amd64 && !__xpv */
1247 }
1248
1249 /*ARGSUSED*/
1250 static void
1251 hat_pcp_teardown(cpu_t *cpu)
 811 hat_vlp_teardown(cpu_t *cpu)
1252 {
1253 #if !defined(__xpv)
 813 #if defined(__amd64) && !defined(__xpv)
1254         struct hat_cpu_info *hci;
1255
1256         if ((hci = cpu->cpu_hat_info) == NULL)
1257                 return;
1258         if (hci->hci_pcp_l2ptes != NULL)
1259                 kmem_free(hci->hci_pcp_l2ptes, MMU_PAGESIZE);
1260         if (hci->hci_pcp_l3ptes != NULL)
1261                 kmem_free(hci->hci_pcp_l3ptes, MMU_PAGESIZE);
1262         if (hci->hci_user_hat != NULL) {
1263                 hat_free_start(hci->hci_user_hat);
1264                 hat_free_end(hci->hci_user_hat);
1265         }
 818         if (hci->hci_vlp_l2ptes)
 819                 kmem_free(hci->hci_vlp_l2ptes, MMU_PAGESIZE);
 820         if (hci->hci_vlp_l3ptes)
 821                 kmem_free(hci->hci_vlp_l3ptes, MMU_PAGESIZE);
1266 #endif
1267 }
______unchanged_portion_omitted_

1276 /*
```

```
 * Finish filling in the kernel hat.
 * Pre fill in all top level kernel page table entries for the kernel's
 * part of the address range.  From this point on we can't use any new
 * kernel large pages if they need PTE's at max_level
 *
 * create the kmap mappings.
 */
void
hat_init_finish(void)
{
        size_t          size;
        uint_t          r = 0;
        uintptr_t       va;
        hat_kernel_range_t *rp;


        /*
         * We are now effectively running on the kernel hat.
         * Clearing use_boot_reserve shuts off using the pre-allocated boot
         * reserve for all HAT allocations.  From here on, the reserves are
         * only used when avoiding recursion in kmem_alloc().
         */
        use_boot_reserve = 0;
        htable_adjust_reserve();

        /*
         * User HATs are initialized with copies of all kernel mappings in
         * higher level page tables. Ensure that those entries exist.
         */
#if defined(__amd64)

        NEXT_HKR(r, 3, kernelbase, 0);
#if defined(__xpv)
        NEXT_HKR(r, 3, HYPERVISOR_VIRT_START, HYPERVISOR_VIRT_END);
#endif

#elif defined(__i386)

#if !defined(__xpv)
        if (mmu.pae_hat) {
                va = kernelbase;
                if ((va & LEVEL_MASK(2)) != va) {
                        va = P2ROUNDUP(va, LEVEL_SIZE(2));
                        NEXT_HKR(r, 1, kernelbase, va);
                }
                if (va != 0)
                        NEXT_HKR(r, 2, va, 0);
        } else
#endif /* __xpv */
                NEXT_HKR(r, 1, kernelbase, 0);

#endif /* __i386 */

        num_kernel_ranges = r;

        /*
         * Create all the kernel pagetables that will have entries
         * shared to user HATs.
         */
        for (r = 0; r < num_kernel_ranges; ++r) {
                rp = &kernel_ranges[r];
                for (va = rp->hkr_start_va; va != rp->hkr_end_va;
                    va += LEVEL_SIZE(rp->hkr_level)) {
                        htable_t *ht;

                        if (IN_HYPERVISOR_VA(va))
```



```
                                continue;

                        /* can/must skip if a page mapping already exists */
                        if (rp->hkr_level <= mmu.max_page_level &&
                            (ht = htable_getpage(kas.a_hat, va, NULL)) !=
                            NULL) {
                                htable_release(ht);
                                continue;
                        }

                        (void) htable_create(kas.a_hat, va, rp->hkr_level - 1,
                            NULL);
                }
        }

        /*
         * 32 bit PAE metal kernels use only 4 of the 512 entries in the
         * page holding the top level pagetable. We use the remainder for
         * the "per CPU" page tables for PCP processes.
 917     * the "per CPU" page tables for VLP processes.
         * Map the top level kernel pagetable into the kernel to make
         * it easy to use bcopy access these tables.
         *
         * PAE is required for the 64-bit kernel which uses this as well to
         * perform the per-CPU pagetables. See the big theory statement.
         */
        if (mmu.pae_hat) {
                pcp_page = vmem_alloc(heap_arena, MMU_PAGESIZE, VM_SLEEP);
                hat_devload(kas.a_hat, (caddr_t)pcp_page, MMU_PAGESIZE,
 922            vlp_page = vmem_alloc(heap_arena, MMU_PAGESIZE, VM_SLEEP);
 923            hat_devload(kas.a_hat, (caddr_t)vlp_page, MMU_PAGESIZE,
                    kas.a_hat->hat_htable->ht_pfn,
#if !defined(__xpv)
                    PROT_WRITE |
#endif
                    PROT_READ | HAT_NOSYNC | HAT_UNORDERED_OK,
                    HAT_LOAD | HAT_LOAD_NOCONSIST);
        }
        hat_pcp_setup(CPU);
 931    hat_vlp_setup(CPU);

        /*
         * Create kmap (cached mappings of kernel PTEs)
         * for 32 bit we map from segmap_start .. ekernelheap
         * for 64 bit we map from segmap_start .. segmap_start + segmapsize;
         */
#if defined(__i386)
        size = (uintptr_t)ekernelheap - segmap_start;
#elif defined(__amd64)
        size = segmapsize;
#endif
        hat_kmap_init((uintptr_t)segmap_start, size);

#if !defined(__xpv)
        ASSERT3U(kas.a_hat->hat_htable->ht_pfn, !=, PFN_INVALID);
        ASSERT3U(kpti_safe_cr3, ==,
            MAKECR3(kas.a_hat->hat_htable->ht_pfn, PCID_KERNEL));
#endif
}

/*
 * On 32 bit PAE mode, PTE's are 64 bits, but ordinary atomic memory references
 * are 32 bit, so for safety we must use atomic_cas_64() to install these.
 */
#ifdef __i386
static void
```

```
reload_pae32(hat_t *hat, cpu_t *cpu)
{
        x86pte_t *src;
        x86pte_t *dest;
        x86pte_t pte;
        int i;

        /*
         * Load the 4 entries of the level 2 page table into this
         * cpu's range of the pcp_page and point cr3 at them.
         * cpu's range of the vlp_page and point cr3 at them.
         */
        ASSERT(mmu.pae_hat);
        src = hat->hat_copied_ptes;
        dest = pcp_page + (cpu->cpu_id + 1) * MAX_COPIED_PTES;
        for (i = 0; i < MAX_COPIED_PTES; ++i) {
        src = hat->hat_vlp_ptes;
        dest = vlp_page + (cpu->cpu_id + 1) * VLP_NUM_PTES;
        for (i = 0; i < VLP_NUM_PTES; ++i) {
                for (;;) {
                        pte = dest[i];
                        if (pte == src[i])
                                break;
                        if (atomic_cas_64(dest + i, pte, src[i]) != src[i])
                                break;
                }
        }
}
#endif

/*
 * Update the PCP data on the CPU cpu to the one on the hat. If this is a 32-bit
 * process, then we must update the L2 pages and then the L3. If this is a
 * 64-bit process then we must update the L3 entries.
 */
static void
hat_pcp_update(cpu_t *cpu, const hat_t *hat)
{
        ASSERT3U(hat->hat_flags & HAT_COPIED, !=, 0);

        if ((hat->hat_flags & HAT_COPIED_32) != 0) {
                const x86pte_t *l2src;
                x86pte_t *l2dst, *l3ptes, *l3uptes;
                /*
                 * This is a 32-bit process. To set this up, we need to do the
                 * following:
                 *
                 *  - Copy the 4 L2 PTEs into the dedicated L2 table
                 *  - Zero the user L3 PTEs in the user and kernel page table
                 *  - Set the first L3 PTE to point to the CPU L2 table
                 */
                l2src = hat->hat_copied_ptes;
                l2dst = cpu->cpu_hat_info->hci_pcp_l2ptes;
                l3ptes = cpu->cpu_hat_info->hci_pcp_l3ptes;
                l3uptes = cpu->cpu_hat_info->hci_user_l3ptes;

                l2dst[0] = l2src[0];
                l2dst[1] = l2src[1];
                l2dst[2] = l2src[2];
                l2dst[3] = l2src[3];

                /*
                 * Make sure to use the mmu to get the number of slots. The
                 * number of PLP entries that this has will always be less as
                 * it's a 32-bit process.
                 */
```



```
                bzero(l3ptes, sizeof (x86pte_t) * mmu.top_level_uslots);
                l3ptes[0] = MAKEPTP(cpu->cpu_hat_info->hci_pcp_l2pfn, 2);
                bzero(l3uptes, sizeof (x86pte_t) * mmu.top_level_uslots);
                l3uptes[0] = MAKEPTP(cpu->cpu_hat_info->hci_pcp_l2pfn, 2);
        } else {
                /*
                 * This is a 64-bit process. To set this up, we need to do the
                 * following:
                 *
                 *  - Zero the 4 L2 PTEs in the CPU structure for safety
                 *  - Copy over the new user L3 PTEs into the kernel page table
                 *  - Copy over the new user L3 PTEs into the user page table
                 */
                ASSERT3S(kpti_enable, ==, 1);
                bzero(cpu->cpu_hat_info->hci_pcp_l2ptes, sizeof (x86pte_t) * 4);
                bcopy(hat->hat_copied_ptes, cpu->cpu_hat_info->hci_pcp_l3ptes,
                    sizeof (x86pte_t) * mmu.top_level_uslots);
                bcopy(hat->hat_copied_ptes, cpu->cpu_hat_info->hci_user_l3ptes,
                    sizeof (x86pte_t) * mmu.top_level_uslots);
        }
}

static void
reset_kpti(struct kpti_frame *fr, uint64_t kcr3, uint64_t ucr3)
{
        ASSERT3U(fr->kf_tr_flag, ==, 0);
#if DEBUG
        if (fr->kf_kernel_cr3 != 0) {
                ASSERT3U(fr->kf_lower_redzone, ==, 0xdeadbeefdeadbeef);
                ASSERT3U(fr->kf_middle_redzone, ==, 0xdeadbeefdeadbeef);
                ASSERT3U(fr->kf_upper_redzone, ==, 0xdeadbeefdeadbeef);
        }
#endif

        bzero(fr, offsetof(struct kpti_frame, kf_kernel_cr3));
        bzero(&fr->kf_unused, sizeof (struct kpti_frame) -
            offsetof(struct kpti_frame, kf_unused));

        fr->kf_kernel_cr3 = kcr3;
        fr->kf_user_cr3 = ucr3;
        fr->kf_tr_ret_rsp = (uintptr_t)&fr->kf_tr_rsp;

        fr->kf_lower_redzone = 0xdeadbeefdeadbeef;
        fr->kf_middle_redzone = 0xdeadbeefdeadbeef;
        fr->kf_upper_redzone = 0xdeadbeefdeadbeef;
}

#ifdef __xpv
static void
hat_switch_xen(hat_t *hat)
{
        struct mmuext_op t[2];
        uint_t retcnt;
        uint_t opcnt = 1;
        uint64_t newcr3;

        ASSERT(!(hat->hat_flags & HAT_COPIED));
        ASSERT(!(getcr4() & CR4_PCIDE));

        newcr3 = MAKECR3((uint64_t)hat->hat_htable->ht_pfn, PCID_NONE);

        t[0].cmd = MMUEXT_NEW_BASEPTR;
        t[0].arg1.mfn = mmu_btop(pa_to_ma(newcr3));

        /*
         * There's an interesting problem here, as to what to actually specify
```

```
1533          * when switching to the kernel hat.  For now we'll reuse the kernel hat
1534          * again.
1535          */
1536         t[1].cmd = MMUEXT_NEW_USER_BASEPTR;
1537         if (hat == kas.a_hat)
1538                 t[1].arg1.mfn = mmu_btop(pa_to_ma(newcr3));
1539         else
1540                 t[1].arg1.mfn = pfn_to_mfn(hat->hat_user_ptable);
1541         ++opcnt;

1543         if (HYPERVISOR_mmuext_op(t, opcnt, &retcnt, DOMID_SELF) < 0)
1544                 panic("HYPERVISOR_mmu_update() failed");
1545         ASSERT(retcnt == opcnt);
1546 }
1547 #endif /* __xpv */

1549 /*
1550  * Switch to a new active hat, maintaining bit masks to track active CPUs.
1551  *
1552  * With KPTI, all our HATs except kas should be using PCP.  Thus, to switch
1553  * HATs, we need to copy over the new user PTEs, then set our trampoline context
1554  * as appropriate.
1555  *
1556  * If lacking PCID, we then load our new cr3, which will flush the TLB: we may
1557  * have established userspace TLB entries via kernel accesses, and these are no
1558  * longer valid.  We have to do this eagerly, as we just deleted this CPU from
1559  * ->hat_cpus, so would no longer see any TLB shootdowns.
1560  *
1561  * With PCID enabled, things get a little more complicated.  We would like to
1562  * keep TLB context around when entering and exiting the kernel, and to do this,
1563  * we partition the TLB into two different spaces:
1564  *
1565  * PCID_KERNEL is defined as zero, and used both by kas and all other address
1566  * spaces while in the kernel (post-trampoline).
1567  *
1568  * PCID_USER is used while in userspace.  Therefore, userspace cannot use any
1569  * lingering PCID_KERNEL entries to kernel addresses it should not be able to
1570  * read.
1571  *
1572  * The trampoline cr3s are set not to invalidate on a mov to %cr3. This means if
1573  * we take a journey through the kernel without switching HATs, we have some
1574  * hope of keeping our TLB state around.
1575  *
1576  * On a hat switch, rather than deal with any necessary flushes on the way out
1577  * of the trampolines, we do them upfront here. If we're switching from kas, we
1578  * shouldn't need any invalidation.
1579  *
1580  * Otherwise, we can have stale userspace entries for both PCID_USER (what
1581  * happened before we move onto the kcr3) and PCID_KERNEL (any subsequent
1582  * userspace accesses such as ddi_copyin()).  Since setcr3() won't do these
1583  * flushes on its own in PCIDE, we'll do a non-flushing load and then
1584  * invalidate everything.
1585  * On the 32-bit PAE hypervisor, %cr3 is a 64-bit value, on metal it
1586  * remains a 32-bit value.
1587  */
1588 void
1589 hat_switch(hat_t *hat)
1590 {
1591         uint64_t        newcr3;
1592         cpu_t *cpu = CPU;
1593         hat_t *old = cpu->cpu_current_hat;

1595         /*
1596          * set up this information first, so we don't miss any cross calls
1597          */
1598         if (old != NULL) {
```



```
1599                 if (old == hat)
1600                         return;
1601                 if (old != kas.a_hat)
1602                         CPUSET_ATOMIC_DEL(old->hat_cpus, cpu->cpu_id);
1603         }

1605         /*
1606          * Add this CPU to the active set for this HAT.
1607          */
1608         if (hat != kas.a_hat) {
1609                 CPUSET_ATOMIC_ADD(hat->hat_cpus, cpu->cpu_id);
1610         }
1611         cpu->cpu_current_hat = hat;

1613 #if defined(__xpv)
1614         hat_switch_xen(hat);
1615 #else
1616         struct hat_cpu_info *info = cpu->cpu_m.mcpu_hat_info;
1617         uint64_t pcide = getcr4() & CR4_PCIDE;
1618         uint64_t kcr3, ucr3;
1619         pfn_t tl_kpfn;
1620         ulong_t flag;
1621         /*
1622          * now go ahead and load cr3
1623          */
1624         if (hat->hat_flags & HAT_VLP) {
1625 #if defined(__amd64)
1626                 x86pte_t *vlpptep = cpu->cpu_hat_info->hci_vlp_l2ptes;

1631         EQUIV(kpti_enable, !mmu.pt_global);

1633         if (hat->hat_flags & HAT_COPIED) {
1634                 hat_pcp_update(cpu, hat);
1635                 tl_kpfn = info->hci_pcp_l3pfn;
1636                 VLP_COPY(hat->hat_vlp_ptes, vlpptep);
1637                 newcr3 = MAKECR3(cpu->cpu_hat_info->hci_vlp_pfn);
1638 #elif defined(__i386)
1639                 reload_pae32(hat, cpu);
1640                 newcr3 = MAKECR3(kas.a_hat->hat_htable->ht_pfn) +
1641                     (cpu->cpu_id + 1) * VLP_SIZE;
1642 #endif
1644         } else {
1645                 IMPLY(kpti_enable, hat == kas.a_hat);
1646                 tl_kpfn = hat->hat_htable->ht_pfn;
1644                 newcr3 = MAKECR3((uint64_t)hat->hat_htable->ht_pfn);
1647         }
1648 #ifdef __xpv
1649         {
1650                 struct mmuext_op t[2];
1651                 uint_t retcnt;
1652                 uint_t opcnt = 1;

1651         if (pcide) {
1652                 ASSERT(kpti_enable);

1654                 kcr3 = MAKECR3(tl_kpfn, PCID_KERNEL) | CR3_NOINVL_BIT;
1655                 ucr3 = MAKECR3(info->hci_user_l3pfn, PCID_USER) |
1656                     CR3_NOINVL_BIT;

1658                 setcr3(kcr3);
1659                 if (old != kas.a_hat)
1660                         mmu_flush_tlb(FLUSH_TLB_ALL, NULL);
1661         } else {
1662                 kcr3 = MAKECR3(tl_kpfn, PCID_NONE);
1663                 ucr3 = kpti_enable ?
1664                     MAKECR3(info->hci_user_l3pfn, PCID_NONE) :
```

```
                    0;

                setcr3(kcr3);
        }

                t[0].cmd = MMUEXT_NEW_BASEPTR;
                t[0].arg1.mfn = mmu_btop(pa_to_ma(newcr3));
#if defined(__amd64)
        /*
         * We will already be taking shootdowns for our new HAT, and as KPTI
         * invpcid emulation needs to use kf_user_cr3, make sure we don't get
         * any cross calls while we're inconsistent.  Note that it's harmless to
         * have a *stale* kf_user_cr3 (we just did a FLUSH_TLB_ALL), but a
         * *zero* kf_user_cr3 is not going to go very well.
                 * There's an interesting problem here, as to what to
                 * actually specify when switching to the kernel hat.
                 * For now we'll reuse the kernel hat again.
         */
        if (pcide)
                flag = intr_clear();
                t[1].cmd = MMUEXT_NEW_USER_BASEPTR;
                if (hat == kas.a_hat)
                        t[1].arg1.mfn = mmu_btop(pa_to_ma(newcr3));
                else
                        t[1].arg1.mfn = pfn_to_mfn(hat->hat_user_ptable);
                ++opcnt;
#endif  /* __amd64 */
                if (HYPERVISOR_mmuext_op(t, opcnt, &retcnt, DOMID_SELF) < 0)
                        panic("HYPERVISOR_mmu_update() failed");
                ASSERT(retcnt == opcnt);

        reset_kpti(&cpu->cpu_m.mcpu_kpti, kcr3, ucr3);
        reset_kpti(&cpu->cpu_m.mcpu_kpti_flt, kcr3, ucr3);
        reset_kpti(&cpu->cpu_m.mcpu_kpti_dbg, kcr3, ucr3);

        if (pcide)
                intr_restore(flag);

#endif /* !__xpv */

        }
#else
        setcr3(newcr3);
#endif
        ASSERT(cpu == CPU);
}
```

_____*unchanged_portion_omitted_*

```
/*
 * This the set of PTE bits for PFN, permissions and caching
 * that are allowed to change on a HAT_LOAD_REMAP
 */
#define PT_REMAP_BITS                                                   \
        (PT_PADDR | PT_NX | PT_WRITABLE | PT_WRITETHRU |                \
        PT_NOCACHE | PT_PAT_4K | PT_PAT_LARGE | PT_IGNORE | PT_REF | PT_MOD)

#define REMAPASSERT(EX) if (!(EX)) panic("hati_pte_map: " #EX)
/*
 * Do the low-level work to get a mapping entered into a HAT's pagetables
 * and in the mapping list of the associated page_t.
 */
static int
hati_pte_map(
        htable_t        *ht,
        uint_t          entry,
        page_t          *pp,
```



```
        x86pte_t        pte,
        int             flags,
        void            *pte_ptr)
{
        hat_t           *hat = ht->ht_hat;
        x86pte_t        old_pte;
        level_t         l = ht->ht_level;
        hment_t         *hm;
        uint_t          is_consist;
        uint_t          is_locked;
        int             rv = 0;

        /*
         * Is this a consistent (ie. need mapping list lock) mapping?
         */
        is_consist = (pp != NULL && (flags & HAT_LOAD_NOCONSIST) == 0);

        /*
         * Track locked mapping count in the htable.  Do this first,
         * as we track locking even if there already is a mapping present.
         */
        is_locked = (flags & HAT_LOAD_LOCK) != 0 && hat != kas.a_hat;
        if (is_locked)
                HTABLE_LOCK_INC(ht);

        /*
         * Acquire the page's mapping list lock and get an hment to use.
         * Note that hment_prepare() might return NULL.
         */
        if (is_consist) {
                x86_hm_enter(pp);
                hm = hment_prepare(ht, entry, pp);
        }

        /*
         * Set the new pte, retrieving the old one at the same time.
         */
        old_pte = x86pte_set(ht, entry, pte, pte_ptr);

        /*
         * Did we get a large page / page table collision?
         */
        if (old_pte == LPAGE_ERROR) {
                if (is_locked)
                        HTABLE_LOCK_DEC(ht);
                rv = -1;
                goto done;
        }

        /*
         * If the mapping didn't change there is nothing more to do.
         */
        if (PTE_EQUIV(pte, old_pte))
                goto done;

        /*
         * Install a new mapping in the page's mapping list
         */
        if (!PTE_ISVALID(old_pte)) {
                if (is_consist) {
                        hment_assign(ht, entry, pp, hm);
                        x86_hm_exit(pp);
                } else {
                        ASSERT(flags & HAT_LOAD_NOCONSIST);
                }
#if defined(__amd64)
```

```
1978                **if (ht->ht_flags & HTABLE_COPIED) {**
1366                *if (ht->ht_flags & HTABLE_VLP) {*
1979                        cpu_t *cpu = CPU;
1980                        **hat_pcp_update(cpu, hat);**
1368                        *x86pte_t *vlpptep = cpu->cpu_hat_info->hci_vlp_l2ptes;*
1369                        *VLP_COPY(hat->hat_vlp_ptes, vlpptep);*
1981                }
1982 #endif
1983                HTABLE_INC(ht->ht_valid_cnt);
1984                PGCNT_INC(hat, l);
1985                return (rv);
1986        }

1988        /*
1989         * Remap's are more complicated:
1990         *  - HAT_LOAD_REMAP must be specified if changing the pfn.
1991         *    We also require that NOCONSIST be specified.
1992         *  - Otherwise only permission or caching bits may change.
1993         */
1994        if (!PTE_ISPAGE(old_pte, l))
1995                panic("non-null/page mapping pte=" FMT_PTE, old_pte);

1997        if (PTE2PFN(old_pte, l) != PTE2PFN(pte, l)) {
1998                REMAPASSERT(flags & HAT_LOAD_REMAP);
1999                REMAPASSERT(flags & HAT_LOAD_NOCONSIST);
2000                REMAPASSERT(PTE_GET(old_pte, PT_SOFTWARE) >= PT_NOCONSIST);
2001                REMAPASSERT(pf_is_memory(PTE2PFN(old_pte, l)) ==
2002                    pf_is_memory(PTE2PFN(pte, l)));
2003                REMAPASSERT(!is_consist);
2004        }

2006        /*
2007         * We only let remaps change the certain bits in the PTE.
2008         */
2009        if (PTE_GET(old_pte, ~PT_REMAP_BITS) != PTE_GET(pte, ~PT_REMAP_BITS))
2010                panic("remap bits changed: old_pte="FMT_PTE", pte="FMT_PTE"\n",
2011                    old_pte, pte);

2013        /*
2014         * We don't create any mapping list entries on a remap, so release
2015         * any allocated hment after we drop the mapping list lock.
2016         */
2017 done:
2018        if (is_consist) {
2019                x86_hm_exit(pp);
2020                if (hm != NULL)
2021                        hment_free(hm);
2022        }
2023        return (rv);
2024 }

2026 /*
2027  * Internal routine to load a single page table entry. This only fails if
2028  * we attempt to overwrite a page table link with a large page.
2029  */
2030 static int
2031 hati_load_common(
2032        hat_t           *hat,
2033        uintptr_t       va,
2034        page_t          *pp,
2035        uint_t          attr,
2036        uint_t          flags,
2037        level_t         level,
2038        pfn_t           pfn)
2039 {
2040        htable_t        *ht;
```



```
2041        uint_t          entry;
2042        x86pte_t        pte;
2043        int             rv = 0;

2045        /*
2046         * The number 16 is arbitrary and here to catch a recursion problem
2047         * early before we blow out the kernel stack.
2048         */
2049        ++curthread->t_hatdepth;
2050        ASSERT(curthread->t_hatdepth < 16);

2052        **ASSERT(hat == kas.a_hat || (hat->hat_flags & HAT_PCP) != 0 ||**
2053            **AS_LOCK_HELD(hat->hat_as));**
1441        *ASSERT(hat == kas.a_hat || AS_LOCK_HELD(hat->hat_as));*

2055        if (flags & HAT_LOAD_SHARE)
2056                hat->hat_flags |= HAT_SHARED;

2058        /*
2059         * Find the page table that maps this page if it already exists.
2060         */
2061        ht = htable_lookup(hat, va, level);

2063        /*
2064         * We must have HAT_LOAD_NOCONSIST if page_t is NULL.
2065         */
2066        if (pp == NULL)
2067                flags |= HAT_LOAD_NOCONSIST;

2069        if (ht == NULL) {
2070                ht = htable_create(hat, va, level, NULL);
2071                ASSERT(ht != NULL);
2072        }
2073        **/***
2074         * **htable_va2entry checks this condition as well, but it won't include**
2075         * **much useful info in the panic. So we do it in advance here to include**
2076         * **all the context.**
2077         **/**
2078        **if (ht->ht_vaddr > va || va > HTABLE_LAST_PAGE(ht)) {**
2079                **panic("hati_load_common: bad htable: va=%p, last page=%p, "**
2080                    **"ht->ht_vaddr=%p, ht->ht_level=%d", (void *)va,**
2081                    **(void *)HTABLE_LAST_PAGE(ht), (void *)ht->ht_vaddr,**
2082                    **(int)ht->ht_level);**
2083        **}**
2084        entry = htable_va2entry(va, ht);

2086        /*
2087         * a bunch of paranoid error checking
2088         */
2089        ASSERT(ht->ht_busy > 0);
1467        *if (ht->ht_vaddr > va || va > HTABLE_LAST_PAGE(ht))*
1468                *panic("hati_load_common: bad htable %p, va %p",*
1469                    *(void *)ht, (void *)va);*
2090        ASSERT(ht->ht_level == level);

2092        /*
2093         * construct the new PTE
2094         */
2095        if (hat == kas.a_hat)
2096                attr &= ~PROT_USER;
2097        pte = hati_mkpte(pfn, attr, level, flags);
2098        if (hat == kas.a_hat && va >= kernelbase)
2099                PTE_SET(pte, mmu.pt_global);

2101        /*
2102         * establish the mapping
```

```
2103          */
2104         rv = hati_pte_map(ht, entry, pp, pte, flags, NULL);

2106         /*
2107          * release the htable and any reserves
2108          */
2109         htable_release(ht);
2110         --curthread->t_hatdepth;
2111         return (rv);
2112 }
_____unchanged_portion_omitted_

2537 #if !defined(__xpv)
2538 /*
2539  * Cross call service routine to demap a range of virtual
2540  * pages on the current CPU or flush all mappings in TLB.
1919  * Cross call service routine to demap a virtual page on
1920  * the current CPU or flush all mappings in TLB.
2541  */
1922 /*ARGSUSED*/
2542 static int
2543 hati_demap_func(xc_arg_t a1, xc_arg_t a2, xc_arg_t a3)
2544 {
2545         _NOTE(ARGUNUSED(a3));
2546         hat_t           *hat = (hat_t *)a1;
2547         tlb_range_t     *range = (tlb_range_t *)a2;
1927         caddr_t addr = (caddr_t)a2;
1928         size_t len = (size_t)a3;

2549         /*
2550          * If the target hat isn't the kernel and this CPU isn't operating
2551          * in the target hat, we can ignore the cross call.
2552          */
2553         if (hat != kas.a_hat && hat != CPU->cpu_current_hat)
2554                 return (0);

2556         if (range->tr_va != DEMAP_ALL_ADDR) {
2557                 mmu_flush_tlb(FLUSH_TLB_RANGE, range);
1937         /*
1938          * For a normal address, we flush a range of contiguous mappings
1939          */
1940         if ((uintptr_t)addr != DEMAP_ALL_ADDR) {
1941                 for (size_t i = 0; i < len; i += MMU_PAGESIZE)
1942                         mmu_tlbflush_entry(addr + i);
2558                 return (0);
2559         }

2561         /*
2562          * We are flushing all of userspace.
1947          * Otherwise we reload cr3 to effect a complete TLB flush.
2563          *
2564          * When using PCP, we first need to update this CPU's idea of the PCP
2565          * PTEs.
1949          * A reload of cr3 on a VLP process also means we must also recopy in
1950          * the pte values from the struct hat
2566          */
2567         if (hat->hat_flags & HAT_COPIED) {
1952         if (hat->hat_flags & HAT_VLP) {
2568 #if defined(__amd64)
2569                 hat_pcp_update(CPU, hat);
1954                 x86pte_t *vlpptep = CPU->cpu_hat_info->hci_vlp_l2ptes;

1956                 VLP_COPY(hat->hat_vlp_ptes, vlpptep);
2570 #elif defined(__i386)
2571                 reload_pae32(hat, CPU);
2572 #endif
```



```
2573         }

2575         mmu_flush_tlb(FLUSH_TLB_NONGLOBAL, NULL);
1961         reload_cr3();
2576         return (0);
2577 }

2579 #define TLBIDLE_CPU_HALTED      (0x1UL)
2580 #define TLBIDLE_INVAL_ALL       (0x2UL)
1965 /*
1966  * Flush all TLB entries, including global (ie. kernel) ones.
1967  */
1968 static void
1969 flush_all_tlb_entries(void)
1970 {
1971         ulong_t cr4 = getcr4();

1973         if (cr4 & CR4_PGE) {
1974                 setcr4(cr4 & ~(ulong_t)CR4_PGE);
1975                 setcr4(cr4);

1977                 /*
1978                  * 32 bit PAE also needs to always reload_cr3()
1979                  */
1980                 if (mmu.max_level == 2)
1981                         reload_cr3();
1982         } else {
1983                 reload_cr3();
1984         }
1985 }

1987 #define TLB_CPU_HALTED  (01ul)
1988 #define TLB_INVAL_ALL   (02ul)
2581 #define CAS_TLB_INFO(cpu, old, new)     \
2582         atomic_cas_ulong((ulong_t *)&(cpu)->cpu_m.mcpu_tlb_info, (old), (new))

2584 /*
2585  * Record that a CPU is going idle
2586  */
2587 void
2588 tlb_going_idle(void)
2589 {
2590         atomic_or_ulong((ulong_t *)&CPU->cpu_m.mcpu_tlb_info,
2591             TLBIDLE_CPU_HALTED);
1998         atomic_or_ulong((ulong_t *)&CPU->cpu_m.mcpu_tlb_info, TLB_CPU_HALTED);
2592 }

2594 /*
2595  * Service a delayed TLB flush if coming out of being idle.
2596  * It will be called from cpu idle notification with interrupt disabled.
2597  */
2598 void
2599 tlb_service(void)
2600 {
2601         ulong_t tlb_info;
2602         ulong_t found;

2604         /*
2605          * We only have to do something if coming out of being idle.
2606          */
2607         tlb_info = CPU->cpu_m.mcpu_tlb_info;
2608         if (tlb_info & TLBIDLE_CPU_HALTED) {
2015         if (tlb_info & TLB_CPU_HALTED) {
2609                 ASSERT(CPU->cpu_current_hat == kas.a_hat);

2611                 /*
```

```
2612                  * Atomic clear and fetch of old state.
2613                  */
2614                 while ((found = CAS_TLB_INFO(CPU, tlb_info, 0)) != tlb_info) {
2615                         ASSERT(found & TLBIDLE_CPU_HALTED);
2022                         ASSERT(found & TLB_CPU_HALTED);
2616                         tlb_info = found;
2617                         SMT_PAUSE();
2618                 }
2619                 if (tlb_info & TLBIDLE_INVAL_ALL)
2620                         mmu_flush_tlb(FLUSH_TLB_ALL, NULL);
2026                 if (tlb_info & TLB_INVAL_ALL)
2027                         flush_all_tlb_entries();
2621         }
2622 }
2623 #endif /* !__xpv */

2625 /*
2626  * Internal routine to do cross calls to invalidate a range of pages on
2627  * all CPUs using a given hat.
2628  */
2629 void
2630 hat_tlb_inval_range(hat_t *hat, tlb_range_t *in_range)
2037 hat_tlb_inval_range(hat_t *hat, uintptr_t va, size_t len)
2631 {
2632         extern int      flushes_require_xcalls; /* from mp_startup.c */
2633         cpuset_t        justme;
2634         cpuset_t        cpus_to_shootdown;
2635         tlb_range_t     range = *in_range;
2636 #ifndef __xpv
2637         cpuset_t        check_cpus;
2638         cpu_t           *cpup;
2639         int             c;
2640 #endif

2642         /*
2643          * If the hat is being destroyed, there are no more users, so
2644          * demap need not do anything.
2645          */
2646         if (hat->hat_flags & HAT_FREEING)
2647                 return;

2649         /*
2650          * If demapping from a shared pagetable, we best demap the
2651          * entire set of user TLBs, since we don't know what addresses
2652          * these were shared at.
2653          */
2654         if (hat->hat_flags & HAT_SHARED) {
2655                 hat = kas.a_hat;
2656                 range.tr_va = DEMAP_ALL_ADDR;
2062                 va = DEMAP_ALL_ADDR;
2657         }

2659         /*
2660          * if not running with multiple CPUs, don't use cross calls
2661          */
2662         if (panicstr || !flushes_require_xcalls) {
2663 #ifdef __xpv
2664                 if (range.tr_va == DEMAP_ALL_ADDR) {
2070                 if (va == DEMAP_ALL_ADDR) {
2665                         xen_flush_tlb();
2666                 } else {
2667                         for (size_t i = 0; i < TLB_RANGE_LEN(&range);
2668                             i += MMU_PAGESIZE) {
2669                                 xen_flush_va((caddr_t)(range.tr_va + i));
2073                         for (size_t i = 0; i < len; i += MMU_PAGESIZE)
2074                                 xen_flush_va((caddr_t)(va + i));
```



```
2670                         }
2671                 }
2672 #else
2673                 (void) hati_demap_func((xc_arg_t)hat, (xc_arg_t)&range, 0);
2077                 (void) hati_demap_func((xc_arg_t)hat,
2078                     (xc_arg_t)va, (xc_arg_t)len);
2674 #endif
2675                 return;
2676         }


2679         /*
2680          * Determine CPUs to shootdown. Kernel changes always do all CPUs.
2681          * Otherwise it's just CPUs currently executing in this hat.
2682          */
2683         kpreempt_disable();
2684         CPUSET_ONLY(justme, CPU->cpu_id);
2685         if (hat == kas.a_hat)
2686                 cpus_to_shootdown = khat_cpuset;
2687         else
2688                 cpus_to_shootdown = hat->hat_cpus;

2690 #ifndef __xpv
2691         /*
2692          * If any CPUs in the set are idle, just request a delayed flush
2693          * and avoid waking them up.
2694          */
2695         check_cpus = cpus_to_shootdown;
2696         for (c = 0; c < NCPU && !CPUSET_ISNULL(check_cpus); ++c) {
2697                 ulong_t tlb_info;

2699                 if (!CPU_IN_SET(check_cpus, c))
2700                         continue;
2701                 CPUSET_DEL(check_cpus, c);
2702                 cpup = cpu[c];
2703                 if (cpup == NULL)
2704                         continue;

2706                 tlb_info = cpup->cpu_m.mcpu_tlb_info;
2707                 while (tlb_info == TLBIDLE_CPU_HALTED) {
2708                         (void) CAS_TLB_INFO(cpup, TLBIDLE_CPU_HALTED,
2709                             TLBIDLE_CPU_HALTED | TLBIDLE_INVAL_ALL);
2112                 while (tlb_info == TLB_CPU_HALTED) {
2113                         (void) CAS_TLB_INFO(cpup, TLB_CPU_HALTED,
2114                             TLB_CPU_HALTED | TLB_INVAL_ALL);
2710                         SMT_PAUSE();
2711                         tlb_info = cpup->cpu_m.mcpu_tlb_info;
2712                 }
2713                 if (tlb_info == (TLBIDLE_CPU_HALTED | TLBIDLE_INVAL_ALL)) {
2118                 if (tlb_info == (TLB_CPU_HALTED | TLB_INVAL_ALL)) {
2714                         HATSTAT_INC(hs_tlb_inval_delayed);
2715                         CPUSET_DEL(cpus_to_shootdown, c);
2716                 }
2717         }
2718 #endif

2720         if (CPUSET_ISNULL(cpus_to_shootdown) ||
2721             CPUSET_ISEQUAL(cpus_to_shootdown, justme)) {

2723 #ifdef __xpv
2724                 if (range.tr_va == DEMAP_ALL_ADDR) {
2129                 if (va == DEMAP_ALL_ADDR) {
2725                         xen_flush_tlb();
2726                 } else {
2727                         for (size_t i = 0; i < TLB_RANGE_LEN(&range);
2728                             i += MMU_PAGESIZE) {
```

```
2729                                xen_flush_va((caddr_t)(range.tr_va + i));
2132                        for (size_t i = 0; i < len; i += MMU_PAGESIZE)
2133                                xen_flush_va((caddr_t)(va + i));
2730                        }
2731                }
2732 #else
2733                (void) hati_demap_func((xc_arg_t)hat, (xc_arg_t)&range, 0);
2136                (void) hati_demap_func((xc_arg_t)hat,
2137                    (xc_arg_t)va, (xc_arg_t)len);
2734 #endif

2736        } else {

2738                CPUSET_ADD(cpus_to_shootdown, CPU->cpu_id);
2739 #ifdef __xpv
2740                if (range.tr_va == DEMAP_ALL_ADDR) {
2144                if (va == DEMAP_ALL_ADDR) {
2741                        xen_gflush_tlb(cpus_to_shootdown);
2742                } else {
2743                        for (size_t i = 0; i < TLB_RANGE_LEN(&range);
2744                            i += MMU_PAGESIZE) {
2745                                xen_gflush_va((caddr_t)(range.tr_va + i),
2147                        for (size_t i = 0; i < len; i += MMU_PAGESIZE) {
2148                                xen_gflush_va((caddr_t)(va + i),
2746                                    cpus_to_shootdown);
2747                        }
2748                }
2749 #else
2750                xc_call((xc_arg_t)hat, (xc_arg_t)&range, 0,
2153                xc_call((xc_arg_t)hat, (xc_arg_t)va, (xc_arg_t)len,
2751                    CPUSET2BV(cpus_to_shootdown), hati_demap_func);
2752 #endif

2754        }
2755        kpreempt_enable();
2756 }

2758 void
2759 hat_tlb_inval(hat_t *hat, uintptr_t va)
2760 {
2761        /*
2762         * Create range for a single page.
2763         */
2764        tlb_range_t range;
2765        range.tr_va = va;
2766        range.tr_cnt = 1; /* one page */
2767        range.tr_level = MIN_PAGE_LEVEL; /* pages are MMU_PAGESIZE */

2769        hat_tlb_inval_range(hat, &range);
2164        hat_tlb_inval_range(hat, va, MMU_PAGESIZE);
2770 }
```

*_____unchanged_portion_omitted_*

```
2935 /*
2331  * Do the callbacks for ranges being unloaded.
2332  */
2333 typedef struct range_info {
2334        uintptr_t       rng_va;
2335        ulong_t         rng_cnt;
2336        level_t         rng_level;
2337 } range_info_t;

2339 /*
2936  * Invalidate the TLB, and perform the callback to the upper level VM system,
2937  * for the specified ranges of contiguous pages.
2938  */
```



```
2939 static void
2940 handle_ranges(hat_t *hat, hat_callback_t *cb, uint_t cnt, tlb_range_t *range)
2344 handle_ranges(hat_t *hat, hat_callback_t *cb, uint_t cnt, range_info_t *range)
2941 {
2942        while (cnt > 0) {
2347                size_t len;

2943                --cnt;
2944                hat_tlb_inval_range(hat, &range[cnt]);
2350                len = range[cnt].rng_cnt << LEVEL_SHIFT(range[cnt].rng_level);
2351                hat_tlb_inval_range(hat, (uintptr_t)range[cnt].rng_va, len);

2946                if (cb != NULL) {
2947                        cb->hcb_start_addr = (caddr_t)range[cnt].tr_va;
2354                        cb->hcb_start_addr = (caddr_t)range[cnt].rng_va;
2948                        cb->hcb_end_addr = cb->hcb_start_addr;
2949                        cb->hcb_end_addr += range[cnt].tr_cnt <<
2950                            LEVEL_SHIFT(range[cnt].tr_level);
2356                        cb->hcb_end_addr += len;
2951                        cb->hcb_function(cb);
2952                }
2953        }
2954 }

2956 /*
2957  * Unload a given range of addresses (has optional callback)
2958  *
2959  * Flags:
2960  * define       HAT_UNLOAD              0x00
2961  * define       HAT_UNLOAD_NOSYNC       0x02
2962  * define       HAT_UNLOAD_UNLOCK       0x04
2963  * define       HAT_UNLOAD_OTHER        0x08 - not used
2964  * define       HAT_UNLOAD_UNMAP        0x10 - same as HAT_UNLOAD
2965  */
2966 #define MAX_UNLOAD_CNT (8)
2967 void
2968 hat_unload_callback(
2969        hat_t           *hat,
2970        caddr_t         addr,
2971        size_t          len,
2972        uint_t          flags,
2973        hat_callback_t  *cb)
2974 {
2975        uintptr_t       vaddr = (uintptr_t)addr;
2976        uintptr_t       eaddr = vaddr + len;
2977        htable_t        *ht = NULL;
2978        uint_t          entry;
2979        uintptr_t       contig_va = (uintptr_t)-1L;
2980        tlb_range_t     r[MAX_UNLOAD_CNT];
2386        range_info_t    r[MAX_UNLOAD_CNT];
2981        uint_t          r_cnt = 0;
2982        x86pte_t        old_pte;

2984        XPV_DISALLOW_MIGRATE();
2985        ASSERT(hat == kas.a_hat || eaddr <= _userlimit);
2986        ASSERT(IS_PAGEALIGNED(vaddr));
2987        ASSERT(IS_PAGEALIGNED(eaddr));

2989        /*
2990         * Special case a single page being unloaded for speed. This happens
2991         * quite frequently, COW faults after a fork() for example.
2992         */
2993        if (cb == NULL && len == MMU_PAGESIZE) {
2994                ht = htable_getpte(hat, vaddr, &entry, &old_pte, 0);
2995                if (ht != NULL) {
2996                        if (PTE_ISVALID(old_pte)) {
```

```
2997                                hat_pte_unmap(ht, entry, flags, old_pte,
2998                                    NULL, B_TRUE);
2999                        }
3000                        htable_release(ht);
3001                }
3002                XPV_ALLOW_MIGRATE();
3003                return;
3004        }

3006        while (vaddr < eaddr) {
3007                old_pte = htable_walk(hat, &ht, &vaddr, eaddr);
3008                if (ht == NULL)
3009                        break;

3011                ASSERT(!IN_VA_HOLE(vaddr));

3013                if (vaddr < (uintptr_t)addr)
3014                        panic("hat_unload_callback(): unmap inside large page");

3016                /*
3017                 * We'll do the call backs for contiguous ranges
3018                 */
3019                if (vaddr != contig_va ||
3020                    (r_cnt > 0 && r[r_cnt - 1].tr_level != ht->ht_level)) {
2426                    (r_cnt > 0 && r[r_cnt - 1].rng_level != ht->ht_level)) {
3021                        if (r_cnt == MAX_UNLOAD_CNT) {
3022                                handle_ranges(hat, cb, r_cnt, r);
3023                                r_cnt = 0;
3024                        }
3025                        r[r_cnt].tr_va = vaddr;
3026                        r[r_cnt].tr_cnt = 0;
3027                        r[r_cnt].tr_level = ht->ht_level;
2431                        r[r_cnt].rng_va = vaddr;
2432                        r[r_cnt].rng_cnt = 0;
2433                        r[r_cnt].rng_level = ht->ht_level;
3028                        ++r_cnt;
3029                }

3031                /*
3032                 * Unload one mapping (for a single page) from the page tables.
3033                 * Note that we do not remove the mapping from the TLB yet,
3034                 * as indicated by the tlb=FALSE argument to hat_pte_unmap().
3035                 * handle_ranges() will clear the TLB entries with one call to
3036                 * hat_tlb_inval_range() per contiguous range.  This is
3037                 * safe because the page can not be reused until the
3038                 * callback is made (or we return).
3039                 */
3040                entry = htable_va2entry(vaddr, ht);
3041                hat_pte_unmap(ht, entry, flags, old_pte, NULL, B_FALSE);
3042                ASSERT(ht->ht_level <= mmu.max_page_level);
3043                vaddr += LEVEL_SIZE(ht->ht_level);
3044                contig_va = vaddr;
3045                ++r[r_cnt - 1].tr_cnt;
2451                ++r[r_cnt - 1].rng_cnt;
3046        }
3047        if (ht)
3048                htable_release(ht);

3050        /*
3051         * handle last range for callbacks
3052         */
3053        if (r_cnt > 0)
3054                handle_ranges(hat, cb, r_cnt, r);
3055        XPV_ALLOW_MIGRATE();
3056 }
```



```
3058 /*
3059  * Invalidate a virtual address translation on a slave CPU during
3060  * panic() dumps.
3061  */
3062 void
3063 hat_flush_range(hat_t *hat, caddr_t va, size_t size)
3064 {
3065         ssize_t sz;
3066         caddr_t endva = va + size;

3068         while (va < endva) {
3069                 sz = hat_getpagesize(hat, va);
3070                 if (sz < 0) {
3071 #ifdef __xpv
3072                         xen_flush_tlb();
3073 #else
3074                         mmu_flush_tlb(FLUSH_TLB_ALL, NULL);
2480                         flush_all_tlb_entries();
3075 #endif
3076                         break;
3077                 }
3078 #ifdef __xpv
3079                 xen_flush_va(va);
3080 #else
3081                 mmu_flush_tlb_kpage((uintptr_t)va);
2487                 mmu_tlbflush_entry(va);
3082 #endif
3083                 va += sz;
3084         }
3085 }
______unchanged_portion_omitted_


3689 /*
3690  * hat_unshare() is similar to hat_unload_callback(), but
3691  * we have to look for empty shared pagetables. Note that
3692  * hat_unshare() is always invoked against an entire segment.
3693  */
3694 /*ARGSUSED*/
3695 void
3696 hat_unshare(hat_t *hat, caddr_t addr, size_t len, uint_t ismszc)
3697 {
3698         uint64_t        vaddr = (uintptr_t)addr;
3699         uintptr_t       eaddr = vaddr + len;
3700         htable_t        *ht = NULL;
3701         uint_t          need_demaps = 0;
3702         int             flags = HAT_UNLOAD_UNMAP;
3703         level_t         l;

3705         ASSERT(hat != kas.a_hat);
3706         ASSERT(eaddr <= _userlimit);
3707         ASSERT(IS_PAGEALIGNED(vaddr));
3708         ASSERT(IS_PAGEALIGNED(eaddr));
3709         XPV_DISALLOW_MIGRATE();

3711         /*
3712          * First go through and remove any shared pagetables.
3713          *
3714          * Note that it's ok to delay the TLB shootdown till the entire range is
3715          * finished, because if hat_pageunload() were to unload a shared
3716          * pagetable page, its hat_tlb_inval() will do a global TLB invalidate.
3717          */
3718         l = mmu.max_page_level;
3719         if (l == mmu.max_level)
3720                 --l;
3721         for (; l >= 0; --l) {
```

```
3722                 for (vaddr = (uintptr_t)addr; vaddr < eaddr;
3723                     vaddr = (vaddr & LEVEL_MASK(l + 1)) + LEVEL_SIZE(l + 1)) {
3724                         ASSERT(!IN_VA_HOLE(vaddr));
3725                         /*
3726                          * find a pagetable that maps the current address
3727                          */
3728                         ht = htable_lookup(hat, vaddr, l);
3729                         if (ht == NULL)
3730                                 continue;
3731                         if (ht->ht_flags & HTABLE_SHARED_PFN) {
3732                                 /*
3733                                  * clear page count, set valid_cnt to 0,
3734                                  * let htable_release() finish the job
3735                                  */
3736                                 hat->hat_ism_pgcnt -= ht->ht_valid_cnt <<
3737                                     (LEVEL_SHIFT(ht->ht_level) - MMU_PAGESHIFT);
3738                                 ht->ht_valid_cnt = 0;
3739                                 need_demaps = 1;
3740                         }
3741                         htable_release(ht);
3742                 }
3743         }

3745         /*
3746          * flush the TLBs - since we're probably dealing with MANY mappings
3747          * we just do a full invalidation.
3153          * we do just one CR3 reload.
3748          */
3749         if (!(hat->hat_flags & HAT_FREEING) && need_demaps)
3750                 hat_tlb_inval(hat, DEMAP_ALL_ADDR);

3752         /*
3753          * Now go back and clean up any unaligned mappings that
3754          * couldn't share pagetables.
3755          */
3756         if (!is_it_dism(hat, addr))
3757                 flags |= HAT_UNLOAD_UNLOCK;
3758         hat_unload(hat, addr, len, flags);
3759         XPV_ALLOW_MIGRATE();
3760 }
```

*unchanged_portion_omitted_*

```
4502 /*
4503  * Release a CPU private mapping for the given address.
4504  * We decrement the htable valid count so it might be destroyed.
4505  */
4506 /*ARGSUSED1*/
4507 void
4508 hat_mempte_release(caddr_t addr, hat_mempte_t pte_pa)
4509 {
4510         htable_t        *ht;

4512         XPV_DISALLOW_MIGRATE();
4513         /*
4514          * invalidate any left over mapping and decrement the htable valid count
4515          */
4516 #ifdef __xpv
4517         if (HYPERVISOR_update_va_mapping((uintptr_t)addr, 0,
4518             UVMF_INVLPG | UVMF_LOCAL))
4519                 panic("HYPERVISOR_update_va_mapping() failed");
4520 #else
4521         {
4522                 x86pte_t *pteptr;

4524                 pteptr = x86pte_mapin(mmu_btop(pte_pa),
4525                     (pte_pa & MMU_PAGEOFFSET) >> mmu.pte_size_shift, NULL);
```



```
4526                 if (mmu.pae_hat)
4527                         *pteptr = 0;
4528                 else
4529                         *(x86pte32_t *)pteptr = 0;
4530                 mmu_flush_tlb_kpage((uintptr_t)addr);
3936                 mmu_tlbflush_entry(addr);
4531                 x86pte_mapout();
4532         }
4533 #endif

4535         ht = htable_getpte(kas.a_hat, ALIGN2PAGE(addr), NULL, NULL, 0);
4536         if (ht == NULL)
4537                 panic("hat_mempte_release(): invalid address");
4538         ASSERT(ht->ht_level == 0);
4539         HTABLE_DEC(ht->ht_valid_cnt);
4540         htable_release(ht);
4541         XPV_ALLOW_MIGRATE();
4542 }

4544 /*
4545  * Apply a temporary CPU private mapping to a page. We flush the TLB only
4546  * on this CPU, so this ought to have been called with preemption disabled.
4547  */
4548 void
4549 hat_mempte_remap(
4550         pfn_t           pfn,
4551         caddr_t         addr,
4552         hat_mempte_t    pte_pa,
4553         uint_t          attr,
4554         uint_t          flags)
4555 {
4556         uintptr_t       va = (uintptr_t)addr;
4557         x86pte_t        pte;

4559         /*
4560          * Remap the given PTE to the new page's PFN. Invalidate only
4561          * on this CPU.
4562          */
4563 #ifdef DEBUG
4564         htable_t        *ht;
4565         uint_t          entry;

4567         ASSERT(IS_PAGEALIGNED(va));
4568         ASSERT(!IN_VA_HOLE(va));
4569         ht = htable_getpte(kas.a_hat, va, &entry, NULL, 0);
4570         ASSERT(ht != NULL);
4571         ASSERT(ht->ht_level == 0);
4572         ASSERT(ht->ht_valid_cnt > 0);
4573         ASSERT(ht->ht_pfn == mmu_btop(pte_pa));
4574         htable_release(ht);
4575 #endif
4576         XPV_DISALLOW_MIGRATE();
4577         pte = hati_mkpte(pfn, attr, 0, flags);
4578 #ifdef __xpv
4579         if (HYPERVISOR_update_va_mapping(va, pte, UVMF_INVLPG | UVMF_LOCAL))
4580                 panic("HYPERVISOR_update_va_mapping() failed");
4581 #else
4582         {
4583                 x86pte_t *pteptr;

4585                 pteptr = x86pte_mapin(mmu_btop(pte_pa),
4586                     (pte_pa & MMU_PAGEOFFSET) >> mmu.pte_size_shift, NULL);
4587                 if (mmu.pae_hat)
4588                         *(x86pte_t *)pteptr = pte;
4589                 else
4590                         *(x86pte32_t *)pteptr = (x86pte32_t)pte;
```

```
4591                 mmu_flush_tlb_kpage((uintptr_t)addr);
3997                 mmu_tlbflush_entry(addr);
4592                 x86pte_mapout();
4593         }
4594 #endif
4595         XPV_ALLOW_MIGRATE();
4596 }
_____unchanged_portion_omitted_

4617 /*
4618  * HAT part of cpu initialization.
4619  */
4620 void
4621 hat_cpu_online(struct cpu *cpup)
4622 {
4623         if (cpup != CPU) {
4624                 x86pte_cpu_init(cpup);
4625                 hat_pcp_setup(cpup);
4031                 hat_vlp_setup(cpup);
4626         }
4627         CPUSET_ATOMIC_ADD(khat_cpuset, cpup->cpu_id);
4628 }

4630 /*
4631  * HAT part of cpu deletion.
4632  * (currently, we only call this after the cpu is safely passivated.)
4633  */
4634 void
4635 hat_cpu_offline(struct cpu *cpup)
4636 {
4637         ASSERT(cpup != CPU);

4639         CPUSET_ATOMIC_DEL(khat_cpuset, cpup->cpu_id);
4640         hat_pcp_teardown(cpup);
4046         hat_vlp_teardown(cpup);
4641         x86pte_cpu_fini(cpup);
4642 }
_____unchanged_portion_omitted_
5086 #endif  /* __xpv */

5088 /*
5089  * Helper function to punch in a mapping that we need with the specified
5090  * attributes.
5091  */
5092 void
5093 hati_cpu_punchin(cpu_t *cpu, uintptr_t va, uint_t attrs)
5094 {
5095         int ret;
5096         pfn_t pfn;
5097         hat_t *cpu_hat = cpu->cpu_hat_info->hci_user_hat;

5099         ASSERT3S(kpti_enable, ==, 1);
5100         ASSERT3P(cpu_hat, !=, NULL);
5101         ASSERT3U(cpu_hat->hat_flags & HAT_PCP, ==, HAT_PCP);
5102         ASSERT3U(va & MMU_PAGEOFFSET, ==, 0);

5104         pfn = hat_getpfnum(kas.a_hat, (caddr_t)va);
5105         VERIFY3U(pfn, !=, PFN_INVALID);

5107         /*
5108          * We purposefully don't try to find the page_t. This means that this
5109          * will be marked PT_NOCONSIST; however, given that this is pretty much
5110          * a static mapping that we're using we should be relatively OK.
5111          */
5112         attrs |= HAT_STORECACHING_OK;
5113         ret = hati_load_common(cpu_hat, va, NULL, attrs, 0, 0, pfn);
```



```
5114         VERIFY3S(ret, ==, 0);
5115 }
```

```
**********************************************************
    9504 Fri Apr  6 17:25:05 2018
new/usr/src/uts/i86pc/vm/hat_i86.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright (c) 2014 by Delphix. All rights reserved.
 * Copyright 2018 Joyent, Inc.
 */

#ifndef _VM_HAT_I86_H
#define _VM_HAT_I86_H


#ifdef  __cplusplus
extern "C" {
#endif

/*
 * VM - Hardware Address Translation management.
 *
 * This file describes the contents of the x86_64 HAT data structures.
 */
#include <sys/types.h>
#include <sys/t_lock.h>
#include <sys/cpuvar.h>
#include <sys/x_call.h>
#include <vm/seg.h>
#include <vm/page.h>
#include <sys/vmparam.h>
#include <sys/vm_machparam.h>
#include <sys/promif.h>
#include <vm/hat_pte.h>
#include <vm/htable.h>
#include <vm/hment.h>

/*
 * The essential data types involved:
 *
 * htable_t     - There is one of these for each page table and it is used
```



```
 *              by the HAT to manage the page table.
 *
 * hment_t      - Links together multiple PTEs to a single page.
 */

/*
 * Maximum number of per-CPU pagetable entries that we'll need to cache in the
 * HAT. See the big theory statement in uts/i86pc/vm/hat_i86.c for more
 * information.
 * VLP processes have a 32 bit address range, so their top level is 2 and
 * with only 4 PTEs in that table.
 */
#if defined(__xpv)
/*
 * The Xen hypervisor does not use per-CPU pagetables (PCP). Define a single
 * struct member for it at least to make life easier and not make the member
 * conditional.
 */
#define MAX_COPIED_PTES 1
#else
/*
 * The 64-bit kernel may have up to 512 PTEs present in it for a given process.
 */
#define MAX_COPIED_PTES 512
#endif  /* __xpv */
#define VLP_LEVEL       (2)
#define VLP_NUM_PTES    (4)
#define VLP_SIZE        (VLP_NUM_PTES * sizeof (x86pte_t))
#define TOP_LEVEL(h)    (((h)->hat_flags & HAT_VLP) ? VLP_LEVEL : mmu.max_level)
#define VLP_COPY(fromptep, toptep) { \
        toptep[0] = fromptep[0]; \
        toptep[1] = fromptep[1]; \
        toptep[2] = fromptep[2]; \
        toptep[3] = fromptep[3]; \
}

#define TOP_LEVEL(h)    (((h)->hat_max_level))

/*
 * The hat struct exists for each address space.
 */
struct hat {
        kmutex_t        hat_mutex;
        struct as       *hat_as;
        uint_t          hat_stats;
        pgcnt_t         hat_pages_mapped[MAX_PAGE_LEVEL + 1];
        pgcnt_t         hat_ism_pgcnt;
        cpuset_t        hat_cpus;
        uint16_t        hat_flags;
        uint8_t         hat_max_level;  /* top level of this HAT */
        uint_t          hat_num_copied; /* Actual num of hat_copied_ptes[] */
        htable_t        *hat_htable;    /* top level htable */
        struct hat      *hat_next;
        struct hat      *hat_prev;
        uint_t          hat_num_hash;   /* number of htable hash buckets */
        htable_t        **hat_ht_hash;  /* htable hash buckets */
        htable_t        *hat_ht_cached; /* cached free htables */
        x86pte_t        hat_copied_ptes[MAX_COPIED_PTES];
        x86pte_t        hat_vlp_ptes[VLP_NUM_PTES];
#if defined(__amd64) && defined(__xpv)
        pfn_t           hat_user_ptable; /* alt top ptable for user mode */
#endif
};
typedef struct hat hat_t;

#define PGCNT_INC(hat, level)   \
```

```
113         atomic_inc_ulong(&(hat)->hat_pages_mapped[level]);
114 #define PGCNT_DEC(hat, level)   \
115         atomic_dec_ulong(&(hat)->hat_pages_mapped[level]);

117 /*
118  * Flags for the hat_flags field. For more information, please see the big
119  * theory statement on the HAT design in uts/i86pc/vm/hat_i86.c.
109  * Flags for the hat_flags field
120  *
121  * HAT_FREEING - set when HAT is being destroyed - mostly used to detect that
122  *       demap()s can be avoided.
123  *
124  * HAT_COPIED - Indicates this HAT is a source for per-cpu page tables: see the
125  *       big comment in hat_i86.c for a description.
114  * HAT_VLP - indicates a 32 bit process has a virtual address range less than
115  *       the hardware's physical address range. (VLP->Virtual Less-than Physical)
116  *       Note - never used on the hypervisor.
126  *
127  * HAT_COPIED_32 - HAT_COPIED, but for an ILP32 process.
128  *
129  * HAT_VICTIM - This is set while a hat is being examined for page table
130  *       stealing and prevents it from being freed.
131  *
132  * HAT_SHARED - The hat has exported it's page tables via hat_share()
133  *
134  * HAT_PINNED - On the hypervisor, indicates the top page table has been pinned.
135  *
136  * HAT_PCP - Used for the per-cpu user page table (i.e. associated with a CPU,
137  *       not a process).
138  */
139 #define HAT_FREEING     (0x0001)
140 #define HAT_VICTIM      (0x0002)
141 #define HAT_SHARED      (0x0004)
142 #define HAT_PINNED      (0x0008)
143 #define HAT_COPIED      (0x0010)
144 #define HAT_COPIED_32   (0x0020)
145 #define HAT_PCP         (0x0040)
126 #define HAT_VLP         (0x0002)
127 #define HAT_VICTIM      (0x0004)
128 #define HAT_SHARED      (0x0008)
129 #define HAT_PINNED      (0x0010)

147 /*
148  * Additional platform attribute for hat_devload() to force no caching.
149  */
150 #define HAT_PLAT_NOCACHE        (0x100000)

152 /*
153  * Simple statistics for the HAT. These are just counters that are
154  * atomically incremented. They can be reset directly from the kernel
155  * debugger.
156  */
157 struct hatstats {
158         ulong_t hs_reap_attempts;
159         ulong_t hs_reaped;
160         ulong_t hs_steals;
161         ulong_t hs_ptable_allocs;
162         ulong_t hs_ptable_frees;
163         ulong_t hs_htable_rgets;        /* allocs from reserve */
164         ulong_t hs_htable_rputs;        /* putbacks to reserve */
165         ulong_t hs_htable_shared;       /* number of htables shared */
166         ulong_t hs_htable_unshared;     /* number of htables unshared */
167         ulong_t hs_hm_alloc;
168         ulong_t hs_hm_free;
169         ulong_t hs_hm_put_reserve;
170         ulong_t hs_hm_get_reserve;
```



```
171         ulong_t hs_hm_steals;
172         ulong_t hs_hm_steal_exam;
173         ulong_t hs_tlb_inval_delayed;
174         ulong_t hs_hat_copied64;
175         ulong_t hs_hat_copied32;
176         ulong_t hs_hat_normal64;
177 };
178 extern struct hatstats hatstat;
179 #ifdef DEBUG
180 #define HATSTAT_INC(x)  (++hatstat.x)
181 #else
182 #define HATSTAT_INC(x)  (0)
183 #endif

185 #if defined(_KERNEL)

187 /*
188  * Useful macro to align hat_XXX() address arguments to a page boundary
189  */
190 #define ALIGN2PAGE(a)           ((uintptr_t)(a) & MMU_PAGEMASK)
191 #define IS_PAGEALIGNED(a)       (((uintptr_t)(a) & MMU_PAGEOFFSET) == 0)

193 extern uint_t   khat_running;   /* set at end of hat_kern_setup() */
194 extern cpuset_t khat_cpuset;    /* cpuset for kernal address demap Xcalls */
195 extern kmutex_t hat_list_lock;
196 extern kcondvar_t hat_list_cv;



200 /*
201  * Interfaces to setup a cpu private mapping (ie. preemption disabled).
202  * The attr and flags arguments are the same as for hat_devload().
203  * setup() must be called once, then any number of calls to remap(),
204  * followed by a final call to release()
205  *
206  * Used by ppcopy(), page_zero(), the memscrubber, and the kernel debugger.
207  */
208 typedef paddr_t hat_mempte_t;                           /* phys addr of PTE */
209 extern hat_mempte_t hat_mempte_setup(caddr_t addr);
210 extern void hat_mempte_remap(pfn_t, caddr_t, hat_mempte_t,
211         uint_t attr, uint_t flags);
212 extern void hat_mempte_release(caddr_t addr, hat_mempte_t);

214 /*
215  * Interfaces to manage which thread has access to htable and hment reserves.
216  * The USE_HAT_RESERVES macro should always be recomputed in full. Its value
217  * (due to curthread) can change after any call into kmem/vmem.
218  */
219 extern uint_t can_steal_post_boot;
220 extern uint_t use_boot_reserve;
221 #define USE_HAT_RESERVES()                                      \
222         (use_boot_reserve || curthread->t_hatdepth > 1 ||       \
223         panicstr != NULL || vmem_is_populator())

225 /*
226  * initialization stuff needed by by startup, mp_startup...
227  */
228 extern void hat_cpu_online(struct cpu *);
229 extern void hat_cpu_offline(struct cpu *);
230 extern void setup_vaddr_for_ppcopy(struct cpu *);
231 extern void teardown_vaddr_for_ppcopy(struct cpu *);
232 extern void clear_boot_mappings(uintptr_t, uintptr_t);

234 /*
235  * magic value to indicate that all TLB entries should be demapped.
236  */
```

```
237 #define DEMAP_ALL_ADDR  (~(uintptr_t)0)

239 /*
240  * not in any include file???
241  */
242 extern void halt(char *fmt);

244 /*
245  * x86 specific routines for use online in setup or i86pc/vm files
246  */
247 extern void hat_kern_alloc(caddr_t segmap_base, size_t segmap_size,
248         caddr_t ekernelheap);
249 extern void hat_kern_setup(void);
231 extern void hat_tlb_inval(struct hat *hat, uintptr_t va);
250 extern void hat_pte_unmap(htable_t *ht, uint_t entry, uint_t flags,
251         x86pte_t old_pte, void *pte_ptr, boolean_t tlb);
252 extern void hat_init_finish(void);
253 extern caddr_t hat_kpm_pfn2va(pfn_t pfn);
254 extern pfn_t hat_kpm_va2pfn(caddr_t);
255 extern page_t *hat_kpm_vaddr2page(caddr_t);
256 extern uintptr_t hat_kernelbase(uintptr_t);
257 extern void hat_kmap_init(uintptr_t base, size_t len);

259 extern hment_t *hati_page_unmap(page_t *pp, htable_t *ht, uint_t entry);

261 extern void mmu_calc_user_slots(void);
262 extern void hat_tlb_inval(struct hat *hat, uintptr_t va);
263 extern void hat_switch(struct hat *hat);
243 #if !defined(__xpv)
244 /*
245  * routines to deal with delayed TLB invalidations for idle CPUs
246  */
247 extern void tlb_going_idle(void);
248 extern void tlb_service(void);
249 #endif

265 #define TLB_RANGE_LEN(r)        ((r)->tr_cnt << LEVEL_SHIFT((r)->tr_level))

267 /*
268  * A range of virtual pages for purposes of demapping.
252  * Hat switch function invoked to load a new context into %cr3
269  */
270 typedef struct tlb_range {
271         uintptr_t tr_va;        /* address of page */
272         ulong_t tr_cnt;         /* number of pages in range */
273         int8_t  tr_level;       /* page table level */
274 } tlb_range_t;
254 extern void hat_switch(struct hat *hat);

276 #if defined(__xpv)

278 #define XPV_DISALLOW_MIGRATE()  xen_block_migrate()
279 #define XPV_ALLOW_MIGRATE()     xen_allow_migrate()

281 #define mmu_flush_tlb_page(va)  mmu_invlpg((caddr_t)va)
282 #define mmu_flush_tlb_kpage(va) mmu_invlpg((caddr_t)va)

256 #ifdef __xpv
284 /*
285  * Interfaces to use around code that maps/unmaps grant table references.
286  */
287 extern void hat_prepare_mapping(hat_t *, caddr_t, uint64_t *);
288 extern void hat_release_mapping(hat_t *, caddr_t);

263 #define XPV_DISALLOW_MIGRATE()  xen_block_migrate()
264 #define XPV_ALLOW_MIGRATE()     xen_allow_migrate()
```



```
290 #else

292 #define XPV_DISALLOW_MIGRATE()  /* nothing */
293 #define XPV_ALLOW_MIGRATE()     /* nothing */

295 #define pfn_is_foreign(pfn)     __lintzero

297 typedef enum flush_tlb_type {
298         FLUSH_TLB_ALL = 1,
299         FLUSH_TLB_NONGLOBAL = 2,
300         FLUSH_TLB_RANGE = 3,
301 } flush_tlb_type_t;
273 #endif

303 extern void mmu_flush_tlb(flush_tlb_type_t, tlb_range_t *);
304 extern void mmu_flush_tlb_kpage(uintptr_t);
305 extern void mmu_flush_tlb_page(uintptr_t);

307 extern void hati_cpu_punchin(cpu_t *cpu, uintptr_t va, uint_t attrs);

309 /*
310  * routines to deal with delayed TLB invalidations for idle CPUs
311  */
312 extern void tlb_going_idle(void);
313 extern void tlb_service(void);

315 #endif /* !__xpv */

317 #endif  /* _KERNEL */

319 #ifdef  __cplusplus
320 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    8256 Fri Apr  6 17:25:05 2018
new/usr/src/uts/i86pc/vm/hat_kdi.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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.
 */

/*
 * HAT interfaces used by the kernel debugger to interact with the VM system.
 * These interfaces are invoked when the world is stopped.  As such, no blocking
 * operations may be performed.
 */

#include <sys/cpuvar.h>
#include <sys/kdi_impl.h>
#include <sys/errno.h>
#include <sys/systm.h>
#include <sys/sysmacros.h>
#include <sys/mman.h>
#include <sys/bootconf.h>
#include <sys/cmn_err.h>
#include <vm/seg_kmem.h>
#include <vm/hat_i86.h>
#if defined(__xpv)
#include <sys/hypervisor.h>
#endif
#include <sys/bootinfo.h>
#include <vm/kboot_mmu.h>
#include <sys/machsystm.h>

/*
 * The debugger needs direct access to the PTE of one page table entry
 * in order to implement vtop and physical read/writes
 */
static uintptr_t hat_kdi_page = 0;      /* vaddr for phsical page accesses */
static uint_t use_kbm = 1;
uint_t hat_kdi_use_pae;                 /* if 0, use x86pte32_t for pte type */
```



```
#if !defined(__xpv)
static x86pte_t *hat_kdi_pte = NULL;    /* vaddr of pte for hat_kdi_page */
#endif

/*
 * Get the address for remapping physical pages during boot
 */
void
hat_boot_kdi_init(void)
{
        hat_kdi_page = (uintptr_t)kbm_push(0);  /* first call gets address... */
}
```

*unchanged_portion_omitted_*

```
#else
#define kdi_mtop(m)     (m)
#define kdi_ptom(p)     (p)
#endif

/*ARGSUSED*/
int
kdi_vtop(uintptr_t va, uint64_t *pap)
{
        uintptr_t vaddr = va;
        size_t  len;
        pfn_t   pfn;
        uint_t  prot;
        int     level;
        x86pte_t pte;
        int     index;

        /*
         * if the mmu struct isn't relevant yet, we need to probe
         * the boot loader's pagetables.
         */
        if (!khat_running) {
                if (kbm_probe(&vaddr, &len, &pfn, &prot) == 0)
                        return (ENOENT);
                if (vaddr > va)
                        return (ENOENT);
                if (vaddr < va)
                        pfn += mmu_btop(va - vaddr);
                *pap = pfn_to_pa(pfn) + (vaddr & MMU_PAGEOFFSET);
                return (0);
        }

        /*
         * We can't go through normal hat routines, so we'll use
         * kdi_pread() to walk the page tables
         */
#if defined(__xpv)
        *pap = pfn_to_pa(CPU->cpu_current_hat->hat_htable->ht_pfn);
#else
        *pap = getcr3_pa();
        *pap = getcr3() & MMU_PAGEMASK;
#endif
        for (level = mmu.max_level; ; --level) {
                index = (va >> LEVEL_SHIFT(level)) & (mmu.ptes_per_table - 1);
                *pap += index << mmu.pte_size_shift;
                pte = 0;
                if (kdi_pread((caddr_t)&pte, mmu.pte_size, *pap, &len) != 0)
                        return (ENOENT);
                if (pte == 0)
                        return (ENOENT);
                if (level > 0 && level <= mmu.max_page_level &&
                    (pte & PT_PAGESIZE)) {
```

```
                        *pap = kdi_mtop(pte & PT_PADDR_LGPG);
                        break;
                } else {
                        *pap = kdi_mtop(pte & PT_PADDR);
                        if (level == 0)
                                break;
                }
        }
        *pap += va & LEVEL_OFFSET(level);
        return (0);
}

static int
kdi_prw(caddr_t buf, size_t nbytes, uint64_t pa, size_t *ncopiedp, int doread)
{
        size_t  ncopied = 0;
        off_t   pgoff;
        size_t  sz;
        caddr_t va;
        caddr_t from;
        caddr_t to;
        x86pte_t pte;

        /*
         * if this is called before any initialization - fail
         */
        if (hat_kdi_page == 0)
                return (EAGAIN);

        while (nbytes > 0) {
                /*
                 * figure out the addresses and construct a minimal PTE
                 */
                pgoff = pa & MMU_PAGEOFFSET;
                sz = MIN(nbytes, MMU_PAGESIZE - pgoff);
                va = (caddr_t)hat_kdi_page + pgoff;
                pte = kdi_ptom(mmu_ptob(mmu_btop(pa))) | PT_VALID;
                if (doread) {
                        from = va;
                        to = buf;
                } else {
                        PTE_SET(pte, PT_WRITABLE);
                        from = buf;
                        to = va;
                }

                /*
                 * map the physical page
                 */
                if (use_kbm)
                        (void) kbm_push(pa);
#if defined(__xpv)
                else
                        (void) HYPERVISOR_update_va_mapping(
                            (uintptr_t)va, pte, UVMF_INVLPG);
#else
                else if (hat_kdi_use_pae)
                        *hat_kdi_pte = pte;
                else
                        *(x86pte32_t *)hat_kdi_pte = pte;
                mmu_flush_tlb_kpage(hat_kdi_page);
                mmu_tlbflush_entry((caddr_t)hat_kdi_page);
#endif

                bcopy(from, to, sz);
```



```
                /*
                 * erase the mapping
                 */
                if (use_kbm)
                        kbm_pop();
#if defined(__xpv)
                else
                        (void) HYPERVISOR_update_va_mapping(
                            (uintptr_t)va, 0, UVMF_INVLPG);
#else
                else if (hat_kdi_use_pae)
                        *hat_kdi_pte = 0;
                else
                        *(x86pte32_t *)hat_kdi_pte = 0;
                mmu_flush_tlb_kpage(hat_kdi_page);
                mmu_tlbflush_entry((caddr_t)hat_kdi_page);
#endif

                buf += sz;
                pa += sz;
                nbytes -= sz;
                ncopied += sz;
        }

        if (ncopied == 0)
                return (ENOENT);

        *ncopiedp = ncopied;
        return (0);
}
______unchanged_portion_omitted_

#if !defined(__xpv)
/*
 * This gets used for flushing the TLB on all the slaves just prior to doing a
 * kdi_prw().  It's unclear why this was originally done, since kdi_prw() itself
 * will flush any lingering hat_kdi_page mappings, but let's presume it was a
 * good idea.
 */
void
kdi_flush_caches(void)
{
        mmu_flush_tlb(FLUSH_TLB_ALL, NULL);
}
#endif

/*
 * Return the number of bytes, relative to the beginning of a given range, that
 * are non-toxic (can be read from and written to with relative impunity).
 */
/*ARGSUSED*/
size_t
kdi_range_is_nontoxic(uintptr_t va, size_t sz, int write)
{
#if defined(__amd64)
        extern uintptr_t toxic_addr;
        extern size_t   toxic_size;

        /*
         * Check 64 bit toxic range.
         */
        if (toxic_addr != 0 &&
            va + sz >= toxic_addr &&
            va < toxic_addr + toxic_size)
                return (va < toxic_addr ? toxic_addr - va : 0);
```

```
335         /*
336          * avoid any Virtual Address hole
337          */
338         if (va + sz >= hole_start && va < hole_end)
339                 return (va < hole_start ? hole_start - va : 0);

341         return (sz);

343 #elif defined(__i386)
344         extern void *device_arena_contains(void *, size_t, size_t *);
345         uintptr_t v;

347         v = (uintptr_t)device_arena_contains((void *)va, sz, NULL);
348         if (v == 0)
349                 return (sz);
350         else if (v <= va)
351                 return (0);
352         else
353                 return (v - va);

355 #endif  /* __i386 */
356 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    9324 Fri Apr  6 17:25:05 2018
new/usr/src/uts/i86pc/vm/hat_pte.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9208 hati_demap_func should take pagesize into account
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Tim Kordas <tim.kordas@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  * Copyright 2018 Joyent, Inc.
  25  */

  27 #ifndef _VM_HAT_PTE_H
  28 #define _VM_HAT_PTE_H

  30 #ifdef  __cplusplus
  31 extern "C" {
  32 #endif

  34 #include <sys/types.h>
  35 #include <sys/mach_mmu.h>

  37 /*
  38  * macros to get/set/clear the PTE fields
  39  */
  40 #define PTE_SET(p, f)   ((p) |= (f))
  41 #define PTE_CLR(p, f)   ((p) &= ~(x86pte_t)(f))
  42 #define PTE_GET(p, f)   ((p) & (f))

  44 /*
  45  * Handy macro to check if a pagetable entry or pointer is valid
  46  */
  47 #define PTE_ISVALID(p)          PTE_GET(p, PT_VALID)

  49 /*
  50  * Does a PTE map a large page.
  51  */
  52 #define PTE_IS_LGPG(p, l)       ((l) > 0 && PTE_GET((p), PT_PAGESIZE))

  54 /*
  55  * does this PTE represent a page (not a pointer to another page table)?
```



```
  56  */
  57 #define PTE_ISPAGE(p, l)        \
  58         (PTE_ISVALID(p) && ((l) == 0 || PTE_GET(p, PT_PAGESIZE)))

  60 /*
  61  * Handy macro to check if 2 PTE's are the same - ignores REF/MOD bits.
  62  * On the 64 bit hypervisor we also have to ignore the high order
  63  * software bits and the global/user bit which are set/cleared
  64  * capriciously (by the hypervisor!)
  65  */
  66 #if defined(__amd64) && defined(__xpv)
  67 #define PT_IGNORE       ((0x7fful << 52) | PT_GLOBAL | PT_USER)
  68 #else
  69 #define PT_IGNORE       (0)
  70 #endif
  71 #define PTE_EQUIV(a, b)  (((a) | (PT_IGNORE | PT_REF | PT_MOD)) == \
  72         ((b) | (PT_IGNORE | PT_REF | PT_MOD)))

  74 /*
  75  * Shorthand for converting a PTE to it's pfn.
  76  */
  77 #define PTE2MFN(p, l)   \
  78         mmu_btop(PTE_GET((p), PTE_IS_LGPG((p), (l)) ? PT_PADDR_LGPG : PT_PADDR))
  79 #ifdef __xpv
  80 #define PTE2PFN(p, l) pte2pfn(p, l)
  81 #else
  82 #define PTE2PFN(p, l) PTE2MFN(p, l)
  83 #endif

  85 #define PT_NX           (0x8000000000000000ull)
  86 #define PT_PADDR        (0x000ffffffffff000ull)
  87 #define PT_PADDR_LGPG   (0x000fffffffffe000ull) /* phys addr for large pages */

  89 /*
  90  * Macros to create a PTP or PTE from the pfn and level
  91  */
  92 #ifdef __xpv

  94 /*
  95  * we use the highest order bit in physical address pfns to mark foreign mfns
  96  */
  97 #ifdef _LP64
  98 #define PFN_IS_FOREIGN_MFN (1ul << 51)
  99 #else
 100 #define PFN_IS_FOREIGN_MFN (1ul << 31)
 101 #endif

 103 #define MAKEPTP(pfn, l) \
 104         (pa_to_ma(pfn_to_pa(pfn)) | mmu.ptp_bits[(l) + 1])
 105 #define MAKEPTE(pfn, l) \
 106         ((pfn & PFN_IS_FOREIGN_MFN) ? \
 107         ((pfn_to_pa(pfn & ~PFN_IS_FOREIGN_MFN) | mmu.pte_bits[l]) | \
 108         PT_FOREIGN | PT_REF | PT_MOD) : \
 109         (pa_to_ma(pfn_to_pa(pfn)) | mmu.pte_bits[l]))
 110 #else
 111 #define MAKEPTP(pfn, l) \
 112         (pfn_to_pa(pfn) | mmu.ptp_bits[(l) + 1])
 113 #define MAKEPTE(pfn, l) \
 114         (pfn_to_pa(pfn) | mmu.pte_bits[l])
 115 #endif

 117 /*
 118  * The idea of "level" refers to the level where the page table is used in the
 119  * the hardware address translation steps. The level values correspond to the
 120  * following names of tables used in AMD/Intel architecture documents:
 121  *
```

```
 *      AMD/INTEL name          Level #
 *      ----------------------  -------
 *      Page Map Level 4           3
 *      Page Directory Pointer     2
 *      Page Directory             1
 *      Page Table                 0
 *
 * The numbering scheme is such that the values of 0 and 1 can correspond to
 * the pagesize codes used for MPSS support. For now the Maximum level at
 * which you can have a large page is a constant, that may change in
 * future processors.
 *
 * The type of "level_t" is signed so that it can be used like:
 *      level_t l;
 *      ...
 *      while (--l >= 0)
 *              ...
 */
#define MAX_NUM_LEVEL           4
#define MAX_PAGE_LEVEL          2
#define MIN_PAGE_LEVEL          0
typedef int8_t level_t;
#define LEVEL_SHIFT(l)  (mmu.level_shift[l])
#define LEVEL_SIZE(l)   (mmu.level_size[l])
#define LEVEL_OFFSET(l) (mmu.level_offset[l])
#define LEVEL_MASK(l)   (mmu.level_mask[l])

/*
 * Macros to:
 * Check for a PFN above 4Gig and 64Gig for 32 bit PAE support
 */
#define PFN_4G          (4ull * (1024 * 1024 * 1024 / MMU_PAGESIZE))
#define PFN_64G         (64ull * (1024 * 1024 * 1024 / MMU_PAGESIZE))
#define PFN_ABOVE4G(pfn) ((pfn) >= PFN_4G)
#define PFN_ABOVE64G(pfn) ((pfn) >= PFN_64G)

/*
 * The CR3 register holds the physical address of the top level page table,
 * along with the current PCID if any.
 * The CR3 register holds the physical address of the top level page table.
 */
#define MAKECR3(pfn, pcid)      (mmu_ptob(pfn) | pcid)
#define MAKECR3(pfn)    mmu_ptob(pfn)

/*
 * HAT/MMU parameters that depend on kernel mode and/or processor type
 */
struct htable;
struct hat_mmu_info {
        x86pte_t pt_nx;         /* either 0 or PT_NX */
        x86pte_t pt_global;     /* either 0 or PT_GLOBAL */

        pfn_t highest_pfn;

        uint_t num_level;       /* number of page table levels in use */
        uint_t max_level;       /* just num_level - 1 */
        uint_t max_page_level;  /* maximum level at which we can map a page */
        uint_t umax_page_level; /* max user page map level */
        uint_t ptes_per_table;  /* # of entries in lower level page tables */
        uint_t top_level_count; /* # of entries in top-level page table */
        uint_t top_level_uslots; /* # of user slots in top-level page table */
        uint_t num_copied_ents; /* # of PCP-copied PTEs to create */
        /* 32-bit versions of values */
        uint_t top_level_uslots32;
        uint_t max_level32;
        uint_t num_copied_ents32;
```



```
        uint_t top_level_count; /* # of entries in top most level page table */

        uint_t hash_cnt;        /* cnt of entries in htable_hash_cache */
        uint_t hat32_hash_cnt;  /* cnt of entries in 32-bit htable_hash_cache */
        uint_t  vlp_hash_cnt;   /* cnt of entries in vlp htable_hash_cache */

        uint_t pae_hat;         /* either 0 or 1 */

        uintptr_t hole_start;   /* start of VA hole (or -1 if none) */
        uintptr_t hole_end;     /* end of VA hole (or 0 if none) */

        struct htable **kmap_htables; /* htables for segmap + 32 bit heap */
        x86pte_t *kmap_ptes;    /* mapping of pagetables that map kmap */
        uintptr_t kmap_addr;    /* start addr of kmap */
        uintptr_t kmap_eaddr;   /* end addr of kmap */

        uint_t pte_size;        /* either 4 or 8 */
        uint_t pte_size_shift;  /* either 2 or 3 */
        x86pte_t ptp_bits[MAX_NUM_LEVEL];       /* bits set for interior PTP */
        x86pte_t pte_bits[MAX_NUM_LEVEL];       /* bits set for leaf PTE */

        /*
         * A range of VA used to window pages in the i86pc/vm code.
         * See PWIN_XXX macros.
         */
        caddr_t pwin_base;
        caddr_t pwin_pte_va;
        paddr_t pwin_pte_pa;

        /*
         * The following tables are equivalent to PAGEXXXXX at different levels
         * in the page table hierarchy.
         */
        uint_t level_shift[MAX_NUM_LEVEL];      /* PAGESHIFT for given level */
        uintptr_t level_size[MAX_NUM_LEVEL];    /* PAGESIZE for given level */
        uintptr_t level_offset[MAX_NUM_LEVEL];  /* PAGEOFFSET for given level */
        uintptr_t level_mask[MAX_NUM_LEVEL];    /* PAGEMASK for given level */
};
```

*unchanged_portion_omitted_*

```
**********************************************************
   60560 Fri Apr  6 17:25:05 2018
new/usr/src/uts/i86pc/vm/htable.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2014 by Delphix. All rights reserved.
 * Copyright 2018 Joyent, Inc.
 * Copyright 2015 Joyent, Inc.
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/kmem.h>
#include <sys/atomic.h>
#include <sys/bitmap.h>
#include <sys/machparam.h>
#include <sys/machsystm.h>
#include <sys/mman.h>
#include <sys/systm.h>
#include <sys/cpuvar.h>
#include <sys/thread.h>
#include <sys/proc.h>
#include <sys/cpu.h>
#include <sys/kmem.h>
#include <sys/disp.h>
#include <sys/vmem.h>
#include <sys/vmsystm.h>
#include <sys/promif.h>
#include <sys/var.h>
#include <sys/x86_archext.h>
#include <sys/archsystm.h>
#include <sys/bootconf.h>
#include <sys/dumphdr.h>
#include <vm/seg_kmem.h>
#include <vm/seg_kpm.h>
#include <vm/hat.h>
#include <vm/hat_i86.h>
#include <sys/cmn_err.h>
#include <sys/panic.h>

#ifdef __xpv
```



```
#include <sys/hypervisor.h>
#include <sys/xpv_panic.h>
#endif

#include <sys/bootinfo.h>
#include <vm/kboot_mmu.h>

static void x86pte_zero(htable_t *dest, uint_t entry, uint_t count);

kmem_cache_t *htable_cache;

/*
 * The variable htable_reserve_amount, rather than HTABLE_RESERVE_AMOUNT,
 * is used in order to facilitate testing of the htable_steal() code.
 * By resetting htable_reserve_amount to a lower value, we can force
 * stealing to occur.  The reserve amount is a guess to get us through boot.
 */
#define HTABLE_RESERVE_AMOUNT   (200)
uint_t htable_reserve_amount = HTABLE_RESERVE_AMOUNT;
kmutex_t htable_reserve_mutex;
uint_t htable_reserve_cnt;
htable_t *htable_reserve_pool;

/*
 * Used to hand test htable_steal().
 */
#ifdef DEBUG
ulong_t force_steal = 0;
ulong_t ptable_cnt = 0;
#endif

/*
 * This variable is so that we can tune this via /etc/system
 * Any value works, but a power of two <= mmu.ptes_per_table is best.
 */
uint_t htable_steal_passes = 8;

/*
 * mutex stuff for access to htable hash
 */
#define NUM_HTABLE_MUTEX 128
kmutex_t htable_mutex[NUM_HTABLE_MUTEX];
#define HTABLE_MUTEX_HASH(h) ((h) & (NUM_HTABLE_MUTEX - 1))

#define HTABLE_ENTER(h) mutex_enter(&htable_mutex[HTABLE_MUTEX_HASH(h)]);
#define HTABLE_EXIT(h)  mutex_exit(&htable_mutex[HTABLE_MUTEX_HASH(h)]);

/*
 * forward declarations
 */
static void link_ptp(htable_t *higher, htable_t *new, uintptr_t vaddr);
static void unlink_ptp(htable_t *higher, htable_t *old, uintptr_t vaddr);
static void htable_free(htable_t *ht);
static x86pte_t *x86pte_access_pagetable(htable_t *ht, uint_t index);
static void x86pte_release_pagetable(htable_t *ht);
static x86pte_t x86pte_cas(htable_t *ht, uint_t entry, x86pte_t old,
        x86pte_t new);

/*
 * A counter to track if we are stealing or reaping htables. When non-zero
 * htable_free() will directly free htables (either to the reserve or kmem)
 * instead of putting them in a hat's htable cache.
 */
uint32_t htable_dont_cache = 0;

/*
```

```
 125  * Track the number of active pagetables, so we can know how many to reap
 126  */
 127 static uint32_t active_ptables = 0;

 129 #ifdef __xpv
 130 /*
 131  * Deal with hypervisor complications.
 132  */
 133 void
 134 xen_flush_va(caddr_t va)
 135 {
 136         struct mmuext_op t;
 137         uint_t count;

 139         if (IN_XPV_PANIC()) {
 140                 mmu_flush_tlb_page((uintptr_t)va);
 140                 mmu_tlbflush_entry((caddr_t)va);
 141         } else {
 142                 t.cmd = MMUEXT_INVLPG_LOCAL;
 143                 t.arg1.linear_addr = (uintptr_t)va;
 144                 if (HYPERVISOR_mmuext_op(&t, 1, &count, DOMID_SELF) < 0)
 145                         panic("HYPERVISOR_mmuext_op() failed");
 146                 ASSERT(count == 1);
 147         }
 148 }

 150 void
 151 xen_gflush_va(caddr_t va, cpuset_t cpus)
 152 {
 153         struct mmuext_op t;
 154         uint_t count;

 156         if (IN_XPV_PANIC()) {
 157                 mmu_flush_tlb_page((uintptr_t)va);
 157                 mmu_tlbflush_entry((caddr_t)va);
 158                 return;
 159         }

 161         t.cmd = MMUEXT_INVLPG_MULTI;
 162         t.arg1.linear_addr = (uintptr_t)va;
 163         /*LINTED: constant in conditional context*/
 164         set_xen_guest_handle(t.arg2.vcpumask, &cpus);
 165         if (HYPERVISOR_mmuext_op(&t, 1, &count, DOMID_SELF) < 0)
 166                 panic("HYPERVISOR_mmuext_op() failed");
 167         ASSERT(count == 1);
 168 }
_____unchanged_portion_omitted_

 563 /*
 564  * This routine steals htables from user processes.  Called by htable_reap
 565  * (reap=TRUE) or htable_alloc (reap=FALSE).
 566  */
 567 static htable_t *
 568 htable_steal(uint_t cnt, boolean_t reap)
 569 {
 570         hat_t           *hat = kas.a_hat;       /* list starts with khat */
 571         htable_t        *list = NULL;
 572         htable_t        *ht;
 573         uint_t          stolen = 0;
 574         uint_t          pass, passes;
 575         uint_t          threshold;

 577         /*
 578          * Limit htable_steal_passes to something reasonable
 579          */
 580         if (htable_steal_passes == 0)
```



```
 581                 htable_steal_passes = 1;
 582         if (htable_steal_passes > mmu.ptes_per_table)
 583                 htable_steal_passes = mmu.ptes_per_table;

 585         /*
 586          * If we're stealing merely as part of kmem reaping (versus stealing
 587          * to assure forward progress), we don't want to actually steal any
 588          * active htables.  (Stealing active htables merely to give memory
 589          * back to the system can inadvertently kick off an htable crime wave
 590          * as active processes repeatedly steal htables from one another,
 591          * plummeting the system into a kind of HAT lawlessness that can
 592          * become so violent as to impede the one thing that can end it:  the
 593          * freeing of memory via ARC reclaim and other means.)  So if we're
 594          * reaping, we limit ourselves to the first pass that steals cached
 595          * htables that aren't in use -- which gives memory back, but averts
 596          * the entire breakdown of social order.
 597          */
 598         passes = reap ? 0 : htable_steal_passes;

 600         /*
 601          * Loop through all user hats. The 1st pass takes cached htables that
 602          * aren't in use. The later passes steal by removing mappings, too.
 603          */
 604         atomic_inc_32(&htable_dont_cache);
 605         for (pass = 0; pass <= passes && stolen < cnt; ++pass) {
 606                 threshold = pass * mmu.ptes_per_table / htable_steal_passes;

 608                 mutex_enter(&hat_list_lock);

 610                 /* skip the first hat (kernel) */
 611                 hat = kas.a_hat->hat_next;
 612                 for (;;) {
 613                         /*
 614                          * Skip any hat that is already being stolen from.
 615                          *
 616                          * We skip SHARED hats, as these are dummy
 617                          * hats that host ISM shared page tables.
 618                          *
 619                          * We also skip if HAT_FREEING because hat_pte_unmap()
 620                          * won't zero out the PTE's. That would lead to hitting
 621                          * stale PTEs either here or under hat_unload() when we
 622                          * steal and unload the same page table in competing
 623                          * threads.
 624                          *
 625                          * We skip HATs that belong to CPUs, to make our lives
 626                          * simpler.
 627                          */
 628                         while (hat != NULL && (hat->hat_flags &
 629                             (HAT_VICTIM | HAT_SHARED | HAT_FREEING |
 630                             HAT_PCP)) != 0) {
 625                         while (hat != NULL &&
 626                             (hat->hat_flags &
 627                             (HAT_VICTIM | HAT_SHARED | HAT_FREEING)) != 0)
 631                                 hat = hat->hat_next;
 632                         }

 634                         if (hat == NULL)
 635                                 break;

 637                         /*
 638                          * Mark the HAT as a stealing victim so that it is
 639                          * not freed from under us, e.g. in as_free()
 640                          */
 641                         hat->hat_flags |= HAT_VICTIM;
 642                         mutex_exit(&hat_list_lock);
```

```
644                         /*
645                          * Take any htables from the hat's cached "free" list.
646                          */
647                         hat_enter(hat);
648                         while ((ht = hat->hat_ht_cached) != NULL &&
649                             stolen < cnt) {
650                                 hat->hat_ht_cached = ht->ht_next;
651                                 ht->ht_next = list;
652                                 list = ht;
653                                 ++stolen;
654                         }
655                         hat_exit(hat);

657                         /*
658                          * Don't steal active htables on first pass.
659                          */
660                         if (pass != 0 && (stolen < cnt))
661                                 htable_steal_active(hat, cnt, threshold,
662                                     &stolen, &list);

664                         /*
665                          * do synchronous teardown for the reap case so that
666                          * we can forget hat; at this time, hat is
667                          * guaranteed to be around because HAT_VICTIM is set
668                          * (see htable_free() for similar code)
669                          */
670                         for (ht = list; (ht) && (reap); ht = ht->ht_next) {
671                                 if (ht->ht_hat == NULL)
672                                         continue;
673                                 ASSERT(ht->ht_hat == hat);
674 #if defined(__xpv) && defined(__amd64)
675                                 ASSERT(!(ht->ht_flags & HTABLE_COPIED));
676                                 if (ht->ht_level == mmu.max_level) {
671                                 if (!(ht->ht_flags & HTABLE_VLP) &&
672                                     ht->ht_level == mmu.max_level) {
677                                         ptable_free(hat->hat_user_ptable);
678                                         hat->hat_user_ptable = PFN_INVALID;
679                                 }
680 #endif
681                                 /*
682                                  * forget the hat
683                                  */
684                                 ht->ht_hat = NULL;
685                         }

687                         mutex_enter(&hat_list_lock);

689                         /*
690                          * Are we finished?
691                          */
692                         if (stolen == cnt) {
693                                 /*
694                                  * Try to spread the pain of stealing,
695                                  * move victim HAT to the end of the HAT list.
696                                  */
697                                 if (pass >= 1 && cnt == 1 &&
698                                     kas.a_hat->hat_prev != hat)
699                                         move_victim(hat);
700                                 /*
701                                  * We are finished
702                                  */
703                         }

705                         /*
706                          * Clear the victim flag, hat can go away now (once
707                          * the lock is dropped)
```



```
708                          */
709                         if (hat->hat_flags & HAT_VICTIM) {
710                                 ASSERT(hat != kas.a_hat);
711                                 hat->hat_flags &= ~HAT_VICTIM;
712                                 cv_broadcast(&hat_list_cv);
713                         }

715                         /* move on to the next hat */
716                         hat = hat->hat_next;
717                 }

719                 mutex_exit(&hat_list_lock);

721         }
722         ASSERT(!MUTEX_HELD(&hat_list_lock));

724         atomic_dec_32(&htable_dont_cache);
725         return (list);
726 }
_____unchanged_portion_omitted_

775 /*
776  * Allocate an htable, stealing one or using the reserve if necessary
777  */
778 static htable_t *
779 htable_alloc(
780         hat_t           *hat,
781         uintptr_t       vaddr,
782         level_t         level,
783         htable_t        *shared)
784 {
785         htable_t        *ht = NULL;
786         uint_t          is_copied;
782         uint_t          is_vlp;
787         uint_t          is_bare = 0;
788         uint_t          need_to_zero = 1;
789         int             kmflags = (can_steal_post_boot ? KM_NOSLEEP : KM_SLEEP);

791         if (level < 0 || level > TOP_LEVEL(hat))
792                 panic("htable_alloc(): level %d out of range\n", level);

794         is_copied = (hat->hat_flags & HAT_COPIED) &&
795             level == hat->hat_max_level;
796         if (is_copied || shared != NULL)
790         is_vlp = (hat->hat_flags & HAT_VLP) && level == VLP_LEVEL;
791         if (is_vlp || shared != NULL)
797                 is_bare = 1;

799         /*
800          * First reuse a cached htable from the hat_ht_cached field, this
801          * avoids unnecessary trips through kmem/page allocators.
802          */
803         if (hat->hat_ht_cached != NULL && !is_bare) {
804                 hat_enter(hat);
805                 ht = hat->hat_ht_cached;
806                 if (ht != NULL) {
807                         hat->hat_ht_cached = ht->ht_next;
808                         need_to_zero = 0;
809                         /* XX64 ASSERT() they're all zero somehow */
810                         ASSERT(ht->ht_pfn != PFN_INVALID);
811                 }
812                 hat_exit(hat);
813         }

815         if (ht == NULL) {
816                 /*
```

```
817                  * Allocate an htable, possibly refilling the reserves.
818                  */
819                 if (USE_HAT_RESERVES()) {
820                         ht = htable_get_reserve();
821                 } else {
822                         /*
823                          * Donate successful htable allocations to the reserve.
824                          */
825                         for (;;) {
826                                 ht = kmem_cache_alloc(htable_cache, kmflags);
827                                 if (ht == NULL)
828                                         break;
829                                 ht->ht_pfn = PFN_INVALID;
830                                 if (USE_HAT_RESERVES() ||
831                                     htable_reserve_cnt >= htable_reserve_amount)
832                                         break;
833                                 htable_put_reserve(ht);
834                         }
835                 }
836
837                 /*
838                  * allocate a page for the hardware page table if needed
839                  */
840                 if (ht != NULL && !is_bare) {
841                         ht->ht_hat = hat;
842                         ht->ht_pfn = ptable_alloc((uintptr_t)ht);
843                         if (ht->ht_pfn == PFN_INVALID) {
844                                 if (USE_HAT_RESERVES())
845                                         htable_put_reserve(ht);
846                                 else
847                                         kmem_cache_free(htable_cache, ht);
848                                 ht = NULL;
849                         }
850                 }
851         }
852
853         /*
854          * If allocations failed, kick off a kmem_reap() and resort to
855          * htable steal(). We may spin here if the system is very low on
856          * memory. If the kernel itself has consumed all memory and kmem_reap()
857          * can't free up anything, then we'll really get stuck here.
858          * That should only happen in a system where the administrator has
859          * misconfigured VM parameters via /etc/system.
860          */
861         while (ht == NULL && can_steal_post_boot) {
862                 kmem_reap();
863                 ht = htable_steal(1, B_FALSE);
864                 HATSTAT_INC(hs_steals);
865
866                 /*
867                  * If we stole for a bare htable, release the pagetable page.
868                  */
869                 if (ht != NULL) {
870                         if (is_bare) {
871                                 ptable_free(ht->ht_pfn);
872                                 ht->ht_pfn = PFN_INVALID;
873 #if defined(__xpv) && defined(__amd64)
874                         /*
875                          * make stolen page table writable again in kpm
876                          */
877                         } else if (kpm_vbase && xen_kpm_page(ht->ht_pfn,
878                             PT_VALID | PT_WRITABLE) < 0) {
879                                 panic("failure making kpm r/w pfn=0x%lx",
880                                     ht->ht_pfn);
881 #endif
882                         }
```



```
883                 }
884         }
885
886         /*
887          * All attempts to allocate or steal failed. This should only happen
888          * if we run out of memory during boot, due perhaps to a huge
889          * boot_archive. At this point there's no way to continue.
890          */
891         if (ht == NULL)
892                 panic("htable_alloc(): couldn't steal\n");
893
894 #if defined(__amd64) && defined(__xpv)
895         /*
896          * Under the 64-bit hypervisor, we have 2 top level page tables.
897          * If this allocation fails, we'll resort to stealing.
898          * We use the stolen page indirectly, by freeing the
899          * stolen htable first.
900          */
901         if (level == mmu.max_level) {
902                 for (;;) {
903                         htable_t *stolen;
904
905                         hat->hat_user_ptable = ptable_alloc((uintptr_t)ht + 1);
906                         if (hat->hat_user_ptable != PFN_INVALID)
907                                 break;
908                         stolen = htable_steal(1, B_FALSE);
909                         if (stolen == NULL)
910                                 panic("2nd steal ptable failed\n");
911                         htable_free(stolen);
912                 }
913                 block_zero_no_xmm(kpm_vbase + pfn_to_pa(hat->hat_user_ptable),
914                     MMU_PAGESIZE);
915         }
916 #endif
917
918         /*
919          * Shared page tables have all entries locked and entries may not
920          * be added or deleted.
921          */
922         ht->ht_flags = 0;
923         if (shared != NULL) {
924                 ASSERT(shared->ht_valid_cnt > 0);
925                 ht->ht_flags |= HTABLE_SHARED_PFN;
926                 ht->ht_pfn = shared->ht_pfn;
927                 ht->ht_lock_cnt = 0;
928                 ht->ht_valid_cnt = 0;           /* updated in hat_share() */
929                 ht->ht_shares = shared;
930                 need_to_zero = 0;
931         } else {
932                 ht->ht_shares = NULL;
933                 ht->ht_lock_cnt = 0;
934                 ht->ht_valid_cnt = 0;
935         }
936
937         /*
938          * setup flags, etc. for copied page tables.
933          * setup flags, etc. for VLP htables
939          */
940         if (is_copied) {
941                 ht->ht_flags |= HTABLE_COPIED;
935         if (is_vlp) {
936                 ht->ht_flags |= HTABLE_VLP;
942                 ASSERT(ht->ht_pfn == PFN_INVALID);
943                 need_to_zero = 0;
944         }
```

```
 946         /*
 947          * fill in the htable
 948          */
 949         ht->ht_hat = hat;
 950         ht->ht_parent = NULL;
 951         ht->ht_vaddr = vaddr;
 952         ht->ht_level = level;
 953         ht->ht_busy = 1;
 954         ht->ht_next = NULL;
 955         ht->ht_prev = NULL;

 957         /*
 958          * Zero out any freshly allocated page table
 959          */
 960         if (need_to_zero)
 961                 x86pte_zero(ht, 0, mmu.ptes_per_table);

 963 #if defined(__amd64) && defined(__xpv)
 964         if (!is_bare && kpm_vbase) {
 965                 (void) xen_kpm_page(ht->ht_pfn, PT_VALID);
 966                 if (level == mmu.max_level)
 967                         (void) xen_kpm_page(hat->hat_user_ptable, PT_VALID);
 968         }
 969 #endif

 971         return (ht);
 972 }

 974 /*
 975  * Free up an htable, either to a hat's cached list, the reserves or
 976  * back to kmem.
 977  */
 978 static void
 979 htable_free(htable_t *ht)
 980 {
 981         hat_t *hat = ht->ht_hat;

 983         /*
 984          * If the process isn't exiting, cache the free htable in the hat
 985          * structure. We always do this for the boot time reserve. We don't
 986          * do this if the hat is exiting or we are stealing/reaping htables.
 987          */
 988         if (hat != NULL &&
 989             !(ht->ht_flags & HTABLE_SHARED_PFN) &&
 990             (use_boot_reserve ||
 991             (!(hat->hat_flags & HAT_FREEING) && !htable_dont_cache))) {
 992                 ASSERT((ht->ht_flags & HTABLE_COPIED) == 0);
 987                 ASSERT((ht->ht_flags & HTABLE_VLP) == 0);
 993                 ASSERT(ht->ht_pfn != PFN_INVALID);
 994                 hat_enter(hat);
 995                 ht->ht_next = hat->hat_ht_cached;
 996                 hat->hat_ht_cached = ht;
 997                 hat_exit(hat);
 998                 return;
 999         }

1001         /*
1002          * If we have a hardware page table, free it.
1003          * We don't free page tables that are accessed by sharing.
1004          */
1005         if (ht->ht_flags & HTABLE_SHARED_PFN) {
1006                 ASSERT(ht->ht_pfn != PFN_INVALID);
1007         } else if (!(ht->ht_flags & HTABLE_COPIED)) {
1002         } else if (!(ht->ht_flags & HTABLE_VLP)) {
1008                 ptable_free(ht->ht_pfn);
1009 #if defined(__amd64) && defined(__xpv)
```



```
1010                 if (ht->ht_level == mmu.max_level && hat != NULL) {
1011                         ptable_free(hat->hat_user_ptable);
1012                         hat->hat_user_ptable = PFN_INVALID;
1013                 }
1014 #endif
1015         }
1016         ht->ht_pfn = PFN_INVALID;

1018         /*
1019          * Free it or put into reserves.
1020          */
1021         if (USE_HAT_RESERVES() || htable_reserve_cnt < htable_reserve_amount) {
1022                 htable_put_reserve(ht);
1023         } else {
1024                 kmem_cache_free(htable_cache, ht);
1025                 htable_adjust_reserve();
1026         }
1027 }
______unchanged_portion_omitted_

1090 /*
1091  * Unlink an entry for a table at vaddr and level out of the existing table
1092  * one level higher. We are always holding the HASH_ENTER() when doing this.
1093  */
1094 static void
1095 unlink_ptp(htable_t *higher, htable_t *old, uintptr_t vaddr)
1096 {
1097         uint_t          entry = htable_va2entry(vaddr, higher);
1098         x86pte_t        expect = MAKEPTP(old->ht_pfn, old->ht_level);
1099         x86pte_t        found;
1100         hat_t           *hat = old->ht_hat;

1102         ASSERT(higher->ht_busy > 0);
1103         ASSERT(higher->ht_valid_cnt > 0);
1104         ASSERT(old->ht_valid_cnt == 0);
1105         found = x86pte_cas(higher, entry, expect, 0);
1106 #ifdef __xpv
1107         /*
1108          * This is weird, but Xen apparently automatically unlinks empty
1109          * pagetables from the upper page table. So allow PTP to be 0 already.
1110          */
1111         if (found != expect && found != 0)
1112 #else
1113         if (found != expect)
1114 #endif
1115                 panic("Bad PTP found=" FMT_PTE ", expected=" FMT_PTE,
1116                     found, expect);

1118         /*
1119          * When a top level PTE changes for a copied htable, we must trigger a
1120          * hat_pcp_update() on all HAT CPUs.
1114          * When a top level VLP page table entry changes, we must issue
1115          * a reload of cr3 on all processors.
1121          *
1122          * If we don't need do do that, then we still have to INVLPG against an
1123          * address covered by the inner page table, as the latest processors
1117          * If we don't need do do that, then we still have to INVLPG against
1118          * an address covered by the inner page table, as the latest processors
1124          * have TLB-like caches for non-leaf page table entries.
1125          */
1126         if (!(hat->hat_flags & HAT_FREEING)) {
1127                 hat_tlb_inval(hat, (higher->ht_flags & HTABLE_COPIED) ?
1122                 hat_tlb_inval(hat, (higher->ht_flags & HTABLE_VLP) ?
1128                     DEMAP_ALL_ADDR : old->ht_vaddr);
1129         }
```

```
1131         HTABLE_DEC(higher->ht_valid_cnt);
1132 }

1134 /*
1135  * Link an entry for a new table at vaddr and level into the existing table
1136  * one level higher. We are always holding the HASH_ENTER() when doing this.
1137  */
1138 static void
1139 link_ptp(htable_t *higher, htable_t *new, uintptr_t vaddr)
1140 {
1141         uint_t          entry = htable_va2entry(vaddr, higher);
1142         x86pte_t        newptp = MAKEPTP(new->ht_pfn, new->ht_level);
1143         x86pte_t        found;

1145         ASSERT(higher->ht_busy > 0);

1147         ASSERT(new->ht_level != mmu.max_level);

1149         HTABLE_INC(higher->ht_valid_cnt);

1151         found = x86pte_cas(higher, entry, 0, newptp);
1152         if ((found & ~PT_REF) != 0)
1153                 panic("HAT: ptp not 0, found=" FMT_PTE, found);

1155         /*
1156          * When a top level PTE changes for a copied htable, we must trigger a
1157          * hat_pcp_update() on all HAT CPUs.
1158          *
1151          * When any top level VLP page table entry changes, we must issue
1152          * a reload of cr3 on all processors using it.
1159          * We also need to do this for the kernel hat on PAE 32 bit kernel.
1160          */
1161         if (
1162 #ifdef __i386
1163             (higher->ht_hat == kas.a_hat &&
1164             higher->ht_level == higher->ht_hat->hat_max_level) ||
1157             (higher->ht_hat == kas.a_hat && higher->ht_level == VLP_LEVEL) ||
1165 #endif
1166             (higher->ht_flags & HTABLE_COPIED))
1159             (higher->ht_flags & HTABLE_VLP))
1167                 hat_tlb_inval(higher->ht_hat, DEMAP_ALL_ADDR);
1168 }
______unchanged_portion_omitted_

1285 /*
1286  * Find the htable for the pagetable at the given level for the given address.
1287  * If found acquires a hold that eventually needs to be htable_release()d
1288  */
1289 htable_t *
1290 htable_lookup(hat_t *hat, uintptr_t vaddr, level_t level)
1291 {
1292         uintptr_t       base;
1293         uint_t          hashval;
1294         htable_t        *ht = NULL;

1296         ASSERT(level >= 0);
1297         ASSERT(level <= TOP_LEVEL(hat));

1299         if (level == TOP_LEVEL(hat)) {
1300 #if defined(__amd64)
1301                 /*
1302                  * 32 bit address spaces on 64 bit kernels need to check
1303                  * for overflow of the 32 bit address space
1304                  */
1305                 if ((hat->hat_flags & HAT_COPIED_32) &&
1306                     vaddr >= ((uint64_t)1 << 32))
```



```
1298                 if ((hat->hat_flags & HAT_VLP) && vaddr >= ((uint64_t)1 << 32))
1307                         return (NULL);
1308 #endif
1309                 base = 0;
1310         } else {
1311                 base = vaddr & LEVEL_MASK(level + 1);
1312         }

1314         hashval = HTABLE_HASH(hat, base, level);
1315         HTABLE_ENTER(hashval);
1316         for (ht = hat->hat_ht_hash[hashval]; ht; ht = ht->ht_next) {
1317                 if (ht->ht_hat == hat &&
1318                     ht->ht_vaddr == base &&
1319                     ht->ht_level == level)
1320                         break;
1321         }
1322         if (ht)
1323                 ++ht->ht_busy;

1325         HTABLE_EXIT(hashval);
1326         return (ht);
1327 }
______unchanged_portion_omitted_
1943 #endif /* __i386 */

1945 /*
1946  * Disable preemption and establish a mapping to the pagetable with the
1947  * given pfn. This is optimized for there case where it's the same
1948  * pfn as we last used referenced from this CPU.
1949  */
1950 static x86pte_t *
1951 x86pte_access_pagetable(htable_t *ht, uint_t index)
1952 {
1953         /*
1954          * HTABLE_COPIED pagetables are contained in the hat_t
1946          * VLP pagetables are contained in the hat_t
1955          */
1956         if (ht->ht_flags & HTABLE_COPIED) {
1957                 ASSERT3U(index, <, ht->ht_hat->hat_num_copied);
1958                 return (PT_INDEX_PTR(ht->ht_hat->hat_copied_ptes, index));
1959         }
1948         if (ht->ht_flags & HTABLE_VLP)
1949                 return (PT_INDEX_PTR(ht->ht_hat->hat_vlp_ptes, index));
1960         return (x86pte_mapin(ht->ht_pfn, index, ht));
1961 }

1963 /*
1964  * map the given pfn into the page table window.
1965  */
1966 /*ARGSUSED*/
1967 x86pte_t *
1968 x86pte_mapin(pfn_t pfn, uint_t index, htable_t *ht)
1969 {
1970         x86pte_t *pteptr;
1971         x86pte_t pte = 0;
1972         x86pte_t newpte;
1973         int x;

1975         ASSERT(pfn != PFN_INVALID);

1977         if (!khat_running) {
1978                 caddr_t va = kbm_remap_window(pfn_to_pa(pfn), 1);
1979                 return (PT_INDEX_PTR(va, index));
1980         }

1982         /*
```

```
1983          * If kpm is available, use it.
1984          */
1985         if (kpm_vbase)
1986                 return (PT_INDEX_PTR(hat_kpm_pfn2va(pfn), index));

1988         /*
1989          * Disable preemption and grab the CPU's hci_mutex
1990          */
1991         kpreempt_disable();

1993         ASSERT(CPU->cpu_hat_info != NULL);
1994         ASSERT(!(getcr4() & CR4_PCIDE));

1996         mutex_enter(&CPU->cpu_hat_info->hci_mutex);
1997         x = PWIN_TABLE(CPU->cpu_id);
1998         pteptr = (x86pte_t *)PWIN_PTE_VA(x);
1999 #ifndef __xpv
2000         if (mmu.pae_hat)
2001                 pte = *pteptr;
2002         else
2003                 pte = *(x86pte32_t *)pteptr;
2004 #endif

2006         newpte = MAKEPTE(pfn, 0) | mmu.pt_global | mmu.pt_nx;

2008         /*
2009          * For hardware we can use a writable mapping.
2010          */
2011 #ifdef __xpv
2012         if (IN_XPV_PANIC())
2013 #endif
2014                 newpte |= PT_WRITABLE;

2016         if (!PTE_EQUIV(newpte, pte)) {

2018 #ifdef __xpv
2019                 if (!IN_XPV_PANIC()) {
2020                         xen_map(newpte, PWIN_VA(x));
2021                 } else
2022 #endif
2023                 {
2024                         XPV_ALLOW_PAGETABLE_UPDATES();
2025                         if (mmu.pae_hat)
2026                                 *pteptr = newpte;
2027                         else
2028                                 *(x86pte32_t *)pteptr = newpte;
2029                         XPV_DISALLOW_PAGETABLE_UPDATES();
2030                         mmu_flush_tlb_kpage((uintptr_t)PWIN_VA(x));
2017                         mmu_tlbflush_entry((caddr_t)(PWIN_VA(x)));
2031                 }
2032         }
2033         return (PT_INDEX_PTR(PWIN_VA(x), index));
2034 }

2036 /*
2037  * Release access to a page table.
2038  */
2039 static void
2040 x86pte_release_pagetable(htable_t *ht)
2041 {
2042         if (ht->ht_flags & HTABLE_COPIED)
2029         /*
2030          * nothing to do for VLP htables
2031          */
2032         if (ht->ht_flags & HTABLE_VLP)
2043                 return;
```



```
2045         x86pte_mapout();
2046 }
_____unchanged_portion_omitted_

2093 /*
2094  * Atomic unconditional set of a page table entry, it returns the previous
2095  * value. For pre-existing mappings if the PFN changes, then we don't care
2096  * about the old pte's REF / MOD bits. If the PFN remains the same, we leave
2097  * the MOD/REF bits unchanged.
2098  *
2099  * If asked to overwrite a link to a lower page table with a large page
2100  * mapping, this routine returns the special value of LPAGE_ERROR. This
2101  * allows the upper HAT layers to retry with a smaller mapping size.
2102  */
2103 x86pte_t
2104 x86pte_set(htable_t *ht, uint_t entry, x86pte_t new, void *ptr)
2105 {
2106         x86pte_t        old;
2107         x86pte_t        prev;
2108         x86pte_t        *ptep;
2109         level_t         l = ht->ht_level;
2110         x86pte_t        pfn_mask = (l != 0) ? PT_PADDR_LGPG : PT_PADDR;
2111         x86pte_t        n;
2112         uintptr_t       addr = htable_e2va(ht, entry);
2113         hat_t           *hat = ht->ht_hat;

2115         ASSERT(new != 0); /* don't use to invalidate a PTE, see x86pte_update */
2116         ASSERT(!(ht->ht_flags & HTABLE_SHARED_PFN));
2117         if (ptr == NULL)
2118                 ptep = x86pte_access_pagetable(ht, entry);
2119         else
2120                 ptep = ptr;

2122         /*
2123          * Install the new PTE. If remapping the same PFN, then
2124          * copy existing REF/MOD bits to new mapping.
2125          */
2126         do {
2127                 prev = GET_PTE(ptep);
2128                 n = new;
2129                 if (PTE_ISVALID(n) && (prev & pfn_mask) == (new & pfn_mask))
2130                         n |= prev & (PT_REF | PT_MOD);

2132                 /*
2133                  * Another thread may have installed this mapping already,
2134                  * flush the local TLB and be done.
2135                  */
2136                 if (prev == n) {
2137                         old = new;
2138 #ifdef __xpv
2139                         if (!IN_XPV_PANIC())
2140                                 xen_flush_va((caddr_t)addr);
2141                         else
2142 #endif
2143                                 mmu_flush_tlb_page(addr);
2133                                 mmu_tlbflush_entry((caddr_t)addr);
2144                         goto done;
2145                 }

2147                 /*
2148                  * Detect if we have a collision of installing a large
2149                  * page mapping where there already is a lower page table.
2150                  */
2151                 if (l > 0 && (prev & PT_VALID) && !(prev & PT_PAGESIZE)) {
2152                         old = LPAGE_ERROR;
```

```
2153                        goto done;
2154                }

2156                XPV_ALLOW_PAGETABLE_UPDATES();
2157                old = CAS_PTE(ptep, prev, n);
2158                XPV_DISALLOW_PAGETABLE_UPDATES();
2159        } while (old != prev);

2161        /*
2162         * Do a TLB demap if needed, ie. the old pte was valid.
2163         *
2164         * Note that a stale TLB writeback to the PTE here either can't happen
2165         * or doesn't matter. The PFN can only change for NOSYNC|NOCONSIST
2166         * mappings, but they were created with REF and MOD already set, so
2167         * no stale writeback will happen.
2168         *
2169         * Segmap is the only place where remaps happen on the same pfn and for
2170         * that we want to preserve the stale REF/MOD bits.
2171         */
2172        if (old & PT_REF)
2173                hat_tlb_inval(hat, addr);

2175 done:
2176        if (ptr == NULL)
2177                x86pte_release_pagetable(ht);
2178        return (old);
2179 }

2181 /*
2182  * Atomic compare and swap of a page table entry. No TLB invalidates are done.
2183  * This is used for links between pagetables of different levels.
2184  * Note we always create these links with dirty/access set, so they should
2185  * never change.
2186  */
2187 x86pte_t
2188 x86pte_cas(htable_t *ht, uint_t entry, x86pte_t old, x86pte_t new)
2189 {
2190        x86pte_t        pte;
2191        x86pte_t        *ptep;
2192 #ifdef __xpv
2193        /*
2194         * We can't use writable pagetables for upper level tables, so fake it.
2195         */
2196        mmu_update_t t[2];
2197        int cnt = 1;
2198        int count;
2199        maddr_t ma;

2201        if (!IN_XPV_PANIC()) {
2202                ASSERT(!(ht->ht_flags & HTABLE_COPIED));
2192                ASSERT(!(ht->ht_flags & HTABLE_VLP));   /* no VLP yet */
2203                ma = pa_to_ma(PT_INDEX_PHYSADDR(pfn_to_pa(ht->ht_pfn), entry));
2204                t[0].ptr = ma | MMU_NORMAL_PT_UPDATE;
2205                t[0].val = new;

2207 #if defined(__amd64)
2208                /*
2209                 * On the 64-bit hypervisor we need to maintain the user mode
2210                 * top page table too.
2211                 */
2212                if (ht->ht_level == mmu.max_level && ht->ht_hat != kas.a_hat) {
2213                        ma = pa_to_ma(PT_INDEX_PHYSADDR(pfn_to_pa(
2214                          ht->ht_hat->hat_user_ptable), entry));
2215                        t[1].ptr = ma | MMU_NORMAL_PT_UPDATE;
2216                        t[1].val = new;
2217                        ++cnt;
```



```
2218                }
2219 #endif  /* __amd64 */

2221                if (HYPERVISOR_mmu_update(t, cnt, &count, DOMID_SELF))
2222                        panic("HYPERVISOR_mmu_update() failed");
2223                ASSERT(count == cnt);
2224                return (old);
2225        }
2226 #endif
2227        ptep = x86pte_access_pagetable(ht, entry);
2228        XPV_ALLOW_PAGETABLE_UPDATES();
2229        pte = CAS_PTE(ptep, old, new);
2230        XPV_DISALLOW_PAGETABLE_UPDATES();
2231        x86pte_release_pagetable(ht);
2232        return (pte);
2233 }
______unchanged_portion_omitted_

2355 #ifndef __xpv
2356 /*
2357  * Copy page tables - this is just a little more complicated than the
2358  * previous routines. Note that it's also not atomic! It also is never
2359  * used for HTABLE_COPIED pagetables.
2349  * used for VLP pagetables.
2360  */
2361 void
2362 x86pte_copy(htable_t *src, htable_t *dest, uint_t entry, uint_t count)
2363 {
2364        caddr_t src_va;
2365        caddr_t dst_va;
2366        size_t size;
2367        x86pte_t *pteptr;
2368        x86pte_t pte;

2370        ASSERT(khat_running);
2371        ASSERT(!(dest->ht_flags & HTABLE_COPIED));
2372        ASSERT(!(src->ht_flags & HTABLE_COPIED));
2361        ASSERT(!(dest->ht_flags & HTABLE_VLP));
2362        ASSERT(!(src->ht_flags & HTABLE_VLP));
2373        ASSERT(!(src->ht_flags & HTABLE_SHARED_PFN));
2374        ASSERT(!(dest->ht_flags & HTABLE_SHARED_PFN));

2376        /*
2377         * Acquire access to the CPU pagetable windows for the dest and source.
2378         */
2379        dst_va = (caddr_t)x86pte_access_pagetable(dest, entry);
2380        if (kpm_vbase) {
2381                src_va = (caddr_t)
2382                    PT_INDEX_PTR(hat_kpm_pfn2va(src->ht_pfn), entry);
2383        } else {
2384                uint_t x = PWIN_SRC(CPU->cpu_id);

2386                ASSERT(!(getcr4() & CR4_PCIDE));

2388                /*
2389                 * Finish defining the src pagetable mapping
2390                 */
2391                src_va = (caddr_t)PT_INDEX_PTR(PWIN_VA(x), entry);
2392                pte = MAKEPTE(src->ht_pfn, 0) | mmu.pt_global | mmu.pt_nx;
2393                pteptr = (x86pte_t *)PWIN_PTE_VA(x);
2394                if (mmu.pae_hat)
2395                        *pteptr = pte;
2396                else
2397                        *(x86pte32_t *)pteptr = pte;
2398                mmu_flush_tlb_kpage((uintptr_t)PWIN_VA(x));
2386                mmu_tlbflush_entry((caddr_t)(PWIN_VA(x)));
```

```
2399         }

2401         /*
2402          * now do the copy
2403          */
2404         size = count << mmu.pte_size_shift;
2405         bcopy(src_va, dst_va, size);

2407         x86pte_release_pagetable(dest);
2408 }
_____unchanged_portion_omitted_
2446 #endif /* __xpv */

2448 /*
2449  * Zero page table entries - Note this doesn't use atomic stores!
2450  */
2451 static void
2452 x86pte_zero(htable_t *dest, uint_t entry, uint_t count)
2453 {
2454         caddr_t dst_va;
2455         size_t size;
2456 #ifdef __xpv
2457         int x;
2458         x86pte_t newpte;
2459 #endif

2461         /*
2462          * Map in the page table to be zeroed.
2463          */
2464         ASSERT(!(dest->ht_flags & HTABLE_SHARED_PFN));
2465         ASSERT(!(dest->ht_flags & HTABLE_COPIED));
2453         ASSERT(!(dest->ht_flags & HTABLE_VLP));

2467         /*
2468          * On the hypervisor we don't use x86pte_access_pagetable() since
2469          * in this case the page is not pinned yet.
2470          */
2471 #ifdef __xpv
2472         if (kpm_vbase == NULL) {
2473                 kpreempt_disable();
2474                 ASSERT(CPU->cpu_hat_info != NULL);
2475                 mutex_enter(&CPU->cpu_hat_info->hci_mutex);
2476                 x = PWIN_TABLE(CPU->cpu_id);
2477                 newpte = MAKEPTE(dest->ht_pfn, 0) | PT_WRITABLE;
2478                 xen_map(newpte, PWIN_VA(x));
2479                 dst_va = (caddr_t)PT_INDEX_PTR(PWIN_VA(x), entry);
2480         } else
2481 #endif
2482                 dst_va = (caddr_t)x86pte_access_pagetable(dest, entry);

2484         size = count << mmu.pte_size_shift;
2485         ASSERT(size > BLOCKZEROALIGN);
2486 #ifdef __i386
2487         if (!is_x86_feature(x86_featureset, X86FSET_SSE2))
2488                 bzero(dst_va, size);
2489         else
2490 #endif
2491                 block_zero_no_xmm(dst_va, size);

2493 #ifdef __xpv
2494         if (kpm_vbase == NULL) {
2495                 xen_map(0, PWIN_VA(x));
2496                 mutex_exit(&CPU->cpu_hat_info->hci_mutex);
2497                 kpreempt_enable();
2498         } else
2499 #endif
```



```
2500                 x86pte_release_pagetable(dest);
2501 }

2503 /*
2504  * Called to ensure that all pagetables are in the system dump
2505  */
2506 void
2507 hat_dump(void)
2508 {
2509         hat_t *hat;
2510         uint_t h;
2511         htable_t *ht;

2513         /*
2514          * Dump all page tables
2515          */
2516         for (hat = kas.a_hat; hat != NULL; hat = hat->hat_next) {
2517                 for (h = 0; h < hat->hat_num_hash; ++h) {
2518                         for (ht = hat->hat_ht_hash[h]; ht; ht = ht->ht_next) {
2519                                 if ((ht->ht_flags & HTABLE_COPIED) == 0)
2507                                 if ((ht->ht_flags & HTABLE_VLP) == 0)
2520                                         dump_page(ht->ht_pfn);
2521                         }
2522                 }
2523         }
2524 }
_____unchanged_portion_omitted_
```

```
**********************************************************
   11423 Fri Apr  6 17:25:06 2018
new/usr/src/uts/i86pc/vm/htable.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2014 by Delphix. All rights reserved.
 * Copyright 2018 Joyent, Inc.
 */

#ifndef _VM_HTABLE_H
#define _VM_HTABLE_H

#ifdef  __cplusplus
extern "C" {
#endif

#if defined(__GNUC__) && defined(_ASM_INLINES) && defined(_KERNEL)
#include <asm/htable.h>
#endif

extern void atomic_andb(uint8_t *addr, uint8_t value);
extern void atomic_orb(uint8_t *addr, uint8_t value);
extern void atomic_inc16(uint16_t *addr);
extern void atomic_dec16(uint16_t *addr);
extern void mmu_tlbflush_entry(caddr_t addr);

/*
 * Each hardware page table has an htable_t describing it.
 *
 * We use a reference counter mechanism to detect when we can free an htable.
 * In the implmentation the reference count is split into 2 separate counters:
 *
 *      ht_busy is a traditional reference count of uses of the htable pointer
 *
 *      ht_valid_cnt is a count of how references are implied by valid PTE/PTP
 *              entries in the pagetable
 *
 * ht_busy is only incremented by htable_lookup() or htable_create()
 * while holding the appropriate hash_table mutex. While installing a new
```



```
 * valid PTE or PTP, in order to increment ht_valid_cnt a thread must have
 * done an htable_lookup() or htable_create() but not the htable_release yet.
 *
 * htable_release(), while holding the mutex, can know that if
 * busy == 1 and valid_cnt == 0, the htable can be free'd.
 *
 * The fields have been ordered to make htable_lookup() fast. Hence,
 * ht_hat, ht_vaddr, ht_level and ht_next need to be clustered together.
 */
struct htable {
        struct htable   *ht_next;       /* forward link for hash table */
        struct hat      *ht_hat;        /* hat this mapping comes from */
        uintptr_t       ht_vaddr;       /* virt addr at start of this table */
        int8_t          ht_level;       /* page table level: 0=4K, 1=2M, ... */
        uint8_t         ht_flags;       /* see below */
        int16_t         ht_busy;        /* implements locking protocol */
        int16_t         ht_valid_cnt;   /* # of valid entries in this table */
        uint32_t        ht_lock_cnt;    /* # of locked entries in this table */
                                        /* never used for kernel hat */
        pfn_t           ht_pfn;         /* pfn of page of the pagetable */
        struct htable   *ht_prev;       /* backward link for hash table */
        struct htable   *ht_parent;     /* htable that points to this htable */
        struct htable   *ht_shares;     /* for HTABLE_SHARED_PFN only */
};
typedef struct htable htable_t;

/*
 * Flags values for htable ht_flags field:
 *
 * HTABLE_COPIED - This is the top level htable of a HAT being used with per-CPU
 *      pagetables.
 * HTABLE_VLP - this is the top level htable of a VLP HAT.
 *
 * HTABLE_SHARED_PFN - this htable had its PFN assigned from sharing another
 *      htable. Used by hat_share() for ISM.
 */
#define HTABLE_COPIED           (0x01)
#define HTABLE_VLP              (0x01)
#define HTABLE_SHARED_PFN       (0x02)

/*
 * The htable hash table hashing function.  The 28 is so that high
 * order bits are include in the hash index to skew the wrap
 * around of addresses. Even though the hash buckets are stored per
 * hat we include the value of hat pointer in the hash function so
 * that the secondary hash for the htable mutex winds up begin different in
 * every address space.
 */
#define HTABLE_HASH(hat, va, lvl)                                       \
        ((((va) >> LEVEL_SHIFT(1)) + ((va) >> 28) + (lvl) +             \
        ((uintptr_t)(hat) >> 4)) & ((hat)->hat_num_hash - 1))

/*
 * Each CPU gets a unique hat_cpu_info structure in cpu_hat_info. For more
 * information on its use and members, see uts/i86pc/vm/hat_i86.c.
 * Each CPU gets a unique hat_cpu_info structure in cpu_hat_info.
 */
struct hat_cpu_info {
        kmutex_t hci_mutex;             /* mutex to ensure sequential usage */
#if defined(__amd64)
        pfn_t   hci_pcp_l3pfn;          /* pfn of hci_pcp_l3ptes */
        pfn_t   hci_pcp_l2pfn;          /* pfn of hci_pcp_l2ptes */
        x86pte_t *hci_pcp_l3ptes;       /* PCP Level==3 pagetable (top) */
        x86pte_t *hci_pcp_l2ptes;       /* PCP Level==2 pagetable */
        struct hat *hci_user_hat;       /* CPU specific HAT */
        pfn_t   hci_user_l3pfn;         /* pfn of hci_user_l3ptes */
```

```
122         x86pte_t *hci_user_l3ptes;      /* PCP User L3 pagetable */
114         pfn_t   hci_vlp_pfn;            /* pfn of hci_vlp_l3ptes */
115         x86pte_t *hci_vlp_l3ptes;       /* VLP Level==3 pagetable (top) */
116         x86pte_t *hci_vlp_l2ptes;       /* VLP Level==2 pagetable */
123 #endif  /* __amd64 */
124 };


127 /*
128  * Compute the last page aligned VA mapped by an htable.
129  *
130  * Given a va and a level, compute the virtual address of the start of the
131  * next page at that level.
132  *
133  * XX64 - The check for the VA hole needs to be better generalized.
134  */
135 #if defined(__amd64)
136 #define HTABLE_NUM_PTES(ht)     (((ht)->ht_flags & HTABLE_COPIED) ? \
137         (((ht)->ht_level == mmu.max_level) ? 512 : 4) : 512)
130 #define HTABLE_NUM_PTES(ht)     (((ht)->ht_flags & HTABLE_VLP) ? 4 : 512)

139 #define HTABLE_LAST_PAGE(ht)                                            \
140         ((ht)->ht_level == mmu.max_level ? ((uintptr_t)0UL - MMU_PAGESIZE) :\
141         ((ht)->ht_vaddr - MMU_PAGESIZE +                                \
142         ((uintptr_t)HTABLE_NUM_PTES(ht) << LEVEL_SHIFT((ht)->ht_level))))

144 #define NEXT_ENTRY_VA(va, l)    \
145         ((va & LEVEL_MASK(l)) + LEVEL_SIZE(l) == mmu.hole_start ?       \
146         mmu.hole_end : (va & LEVEL_MASK(l)) + LEVEL_SIZE(l))

148 #elif defined(__i386)

150 #define HTABLE_NUM_PTES(ht)     \
151         (!mmu.pae_hat ? 1024 : ((ht)->ht_level == 2 ? 4 : 512))

153 #define HTABLE_LAST_PAGE(ht)    ((ht)->ht_vaddr - MMU_PAGESIZE + \
154         ((uintptr_t)HTABLE_NUM_PTES(ht) << LEVEL_SHIFT((ht)->ht_level)))

156 #define NEXT_ENTRY_VA(va, l) ((va & LEVEL_MASK(l)) + LEVEL_SIZE(l))

158 #endif

160 #if defined(_KERNEL)

162 /*
163  * initialization function called from hat_init()
164  */
165 extern void htable_init(void);

167 /*
168  * Functions to lookup, or "lookup and create", the htable corresponding
169  * to the virtual address "vaddr"  in the "hat" at the given "level" of
170  * page tables. htable_lookup() may return NULL if no such entry exists.
171  *
172  * On return the given htable is marked busy (a shared lock) - this prevents
173  * the htable from being stolen or freed) until htable_release() is called.
174  *
175  * If kalloc_flag is set on an htable_create() we can't call kmem allocation
176  * routines for this htable, since it's for the kernel hat itself.
177  *
178  * htable_acquire() is used when an htable pointer has been extracted from
179  * an hment and we need to get a reference to the htable.
180  */
181 extern htable_t *htable_lookup(struct hat *hat, uintptr_t vaddr, level_t level);
182 extern htable_t *htable_create(struct hat *hat, uintptr_t vaddr, level_t level,
183         htable_t *shared);
```



```
184 extern void htable_acquire(htable_t *);

186 extern void htable_release(htable_t *ht);
187 extern void htable_destroy(htable_t *ht);

189 /*
190  * Code to free all remaining htables for a hat. Called after the hat is no
191  * longer in use by any thread.
192  */
193 extern void htable_purge_hat(struct hat *hat);

195 /*
196  * Find the htable, page table entry index, and PTE of the given virtual
197  * address.  If not found returns NULL. When found, returns the htable_t *,
198  * sets entry, and has a hold on the htable.
199  */
200 extern htable_t *htable_getpte(struct hat *, uintptr_t, uint_t *, x86pte_t *,
201         level_t);

203 /*
204  * Similar to hat_getpte(), except that this only succeeds if a valid
205  * page mapping is present.
206  */
207 extern htable_t *htable_getpage(struct hat *hat, uintptr_t va, uint_t *entry);

209 /*
210  * Called to allocate initial/additional htables for reserve.
211  */
212 extern void htable_initial_reserve(uint_t);
213 extern void htable_reserve(uint_t);

215 /*
216  * Used to readjust the htable reserve after the reserve list has been used.
217  * Also called after boot to release left over boot reserves.
218  */
219 extern void htable_adjust_reserve(void);

221 /*
222  * return number of bytes mapped by all the htables in a given hat
223  */
224 extern size_t htable_mapped(struct hat *);


227 /*
228  * Attach initial pagetables as htables
229  */
230 extern void htable_attach(struct hat *, uintptr_t, level_t, struct htable *,
231     pfn_t);

233 /*
234  * Routine to find the next populated htable at or above a given virtual
235  * address. Can specify an upper limit, or HTABLE_WALK_TO_END to indicate
236  * that it should search the entire address space.  Similar to
237  * hat_getpte(), but used for walking through address ranges. It can be
238  * used like this:
239  *
240  *      va = ...
241  *      ht = NULL;
242  *      while (va < end_va) {
243  *              pte = htable_walk(hat, &ht, &va, end_va);
244  *              if (!pte)
245  *                      break;
246  *
247  *              ... code to operate on page at va ...
248  *
249  *              va += LEVEL_SIZE(ht->ht_level);
```

```
 *      }
 *      if (ht)
 *              htable_release(ht);
 *
 */
extern x86pte_t htable_walk(struct hat *hat, htable_t **ht, uintptr_t *va,
        uintptr_t eaddr);

#define HTABLE_WALK_TO_END ((uintptr_t)-1)

/*
 * Utilities convert between virtual addresses and page table entry indeces.
 */
extern uint_t htable_va2entry(uintptr_t va, htable_t *ht);
extern uintptr_t htable_e2va(htable_t *ht, uint_t entry);

/*
 * Interfaces that provide access to page table entries via the htable.
 *
 * Note that all accesses except x86pte_copy() and x86pte_zero() are atomic.
 */
extern void     x86pte_cpu_init(cpu_t *);
extern void     x86pte_cpu_fini(cpu_t *);

extern x86pte_t x86pte_get(htable_t *, uint_t entry);

/*
 * x86pte_set returns LPAGE_ERROR if it's asked to overwrite a page table
 * link with a large page mapping.
 */
#define LPAGE_ERROR (-(x86pte_t)1)
extern x86pte_t x86pte_set(htable_t *, uint_t entry, x86pte_t new, void *);

extern x86pte_t x86pte_inval(htable_t *ht, uint_t entry,
        x86pte_t old, x86pte_t *ptr, boolean_t tlb);

extern x86pte_t x86pte_update(htable_t *ht, uint_t entry,
        x86pte_t old, x86pte_t new);

extern void     x86pte_copy(htable_t *src, htable_t *dest, uint_t entry,
        uint_t cnt);

/*
 * access to a pagetable knowing only the pfn
 */
extern x86pte_t *x86pte_mapin(pfn_t, uint_t, htable_t *);
extern void x86pte_mapout(void);

/*
 * these are actually inlines for "lock; incw", "lock; decw", etc. instructions.
 */
#define HTABLE_INC(x)   atomic_inc16((uint16_t *)&x)
#define HTABLE_DEC(x)   atomic_dec16((uint16_t *)&x)
#define HTABLE_LOCK_INC(ht)     atomic_inc_32(&(ht)->ht_lock_cnt)
#define HTABLE_LOCK_DEC(ht)     atomic_dec_32(&(ht)->ht_lock_cnt)

#ifdef __xpv
extern void xen_flush_va(caddr_t va);
extern void xen_gflush_va(caddr_t va, cpuset_t);
extern void xen_flush_tlb(void);
extern void xen_gflush_tlb(cpuset_t);
extern void xen_pin(pfn_t, level_t);
extern void xen_unpin(pfn_t);
extern int xen_kpm_page(pfn_t, uint_t);

/*
```



```
 * The hypervisor maps all page tables into our address space read-only.
 * Under normal circumstances, the hypervisor then handles all updates to
 * the page tables underneath the covers for us.  However, when we are
 * trying to dump core after a hypervisor panic, the hypervisor is no
 * longer available to do these updates.  To work around the protection
 * problem, we simply disable write-protect checking for the duration of a
 * pagetable update operation.
 */
#define XPV_ALLOW_PAGETABLE_UPDATES()                                   \
        {                                                               \
                if (IN_XPV_PANIC())                                     \
                        setcr0((getcr0() & ~CR0_WP) & 0xffffffff);      \
        }
#define XPV_DISALLOW_PAGETABLE_UPDATES()                                \
        {                                                               \
                if (IN_XPV_PANIC() > 0)                                 \
                        setcr0((getcr0() | CR0_WP) & 0xffffffff);       \
        }

#else /* __xpv */

#define XPV_ALLOW_PAGETABLE_UPDATES()
#define XPV_DISALLOW_PAGETABLE_UPDATES()

#endif

#endif  /* _KERNEL */


#ifdef  __cplusplus
}
_____unchanged_portion_omitted_
```

```
**********************************************************
   15116 Fri Apr  6 17:25:06 2018
new/usr/src/uts/i86pc/vm/i86_mmu.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

#include <sys/t_lock.h>
#include <sys/memlist.h>
#include <sys/cpuvar.h>
#include <sys/vmem.h>
#include <sys/mman.h>
#include <sys/vm.h>
#include <sys/kmem.h>
#include <sys/cmn_err.h>
#include <sys/debug.h>
#include <sys/vm_machparam.h>
#include <sys/tss.h>
#include <sys/vnode.h>
#include <vm/hat.h>
#include <vm/anon.h>
#include <vm/as.h>
#include <vm/page.h>
#include <vm/seg.h>
#include <vm/seg_kmem.h>
#include <vm/seg_map.h>
#include <vm/hat_i86.h>
#include <sys/promif.h>
#include <sys/x86_archext.h>
#include <sys/systm.h>
#include <sys/archsystm.h>
#include <sys/sunddi.h>
#include <sys/ddidmareq.h>
#include <sys/controlregs.h>
#include <sys/reboot.h>
#include <sys/kdi.h>
#include <sys/bootconf.h>
#include <sys/bootsvcs.h>
#include <sys/bootinfo.h>
```



```
#include <vm/kboot_mmu.h>

#ifdef __xpv
#include <sys/hypervisor.h>
#endif

#define ON_USER_HAT(cpu) \
        ((cpu)->cpu_m.mcpu_current_hat != NULL && \
        (cpu)->cpu_m.mcpu_current_hat != kas.a_hat)
caddr_t
i86devmap(pfn_t pf, pgcnt_t pgcnt, uint_t prot)
{
        caddr_t addr;
        caddr_t addr1;
        page_t *pp;

        addr1 = addr = vmem_alloc(heap_arena, mmu_ptob(pgcnt), VM_SLEEP);

        for (; pgcnt != 0; addr += MMU_PAGESIZE, ++pf, --pgcnt) {
                pp = page_numtopp_nolock(pf);
                if (pp == NULL) {
                        hat_devload(kas.a_hat, addr, MMU_PAGESIZE, pf,
                            prot | HAT_NOSYNC, HAT_LOAD_LOCK);
                } else {
                        hat_memload(kas.a_hat, addr, pp,
                            prot | HAT_NOSYNC, HAT_LOAD_LOCK);
                }
        }

        return (addr1);
}

/*
 * This routine is like page_numtopp, but accepts only free pages, which
 * it allocates (unfrees) and returns with the exclusive lock held.
 * It is used by machdep.c/dma_init() to find contiguous free pages.
 *
 * XXX this and some others should probably be in vm_machdep.c
 */
page_t *
page_numtopp_alloc(pfn_t pfnum)
{
        page_t *pp;

retry:
        pp = page_numtopp_nolock(pfnum);
        if (pp == NULL) {
                return (NULL);
        }

        if (!page_trylock(pp, SE_EXCL)) {
                return (NULL);
        }

        if (page_pptonum(pp) != pfnum) {
                page_unlock(pp);
                goto retry;
        }

        if (!PP_ISFREE(pp)) {
                page_unlock(pp);
                return (NULL);
        }
        if (pp->p_szc) {
                page_demote_free_pages(pp);
                page_unlock(pp);
```

```
121                 goto retry;
122         }

124         /* If associated with a vnode, destroy mappings */

126         if (pp->p_vnode) {

128                 page_destroy_free(pp);

130                 if (!page_lock(pp, SE_EXCL, (kmutex_t *)NULL, P_NO_RECLAIM)) {
131                         return (NULL);
132                 }

134                 if (page_pptonum(pp) != pfnum) {
135                         page_unlock(pp);
136                         goto retry;
137                 }
138         }

140         if (!PP_ISFREE(pp)) {
141                 page_unlock(pp);
142                 return (NULL);
143         }

145         if (!page_reclaim(pp, (kmutex_t *)NULL))
146                 return (NULL);

148         return (pp);
149 }

151 /*
 71  * Flag is not set early in boot. Once it is set we are no longer
 72  * using boot's page tables.
 73  */
 74 uint_t khat_running = 0;

 76 /*
 77  * This procedure is callable only while the boot loader is in charge of the
 78  * MMU. It assumes that PA == VA for page table pointers.  It doesn't live in
 79  * kboot_mmu.c since it's used from common code.
 80  */
 81 pfn_t
 82 va_to_pfn(void *vaddr)
 83 {
 84         uintptr_t       des_va = ALIGN2PAGE(vaddr);
 85         uintptr_t       va = des_va;
 86         size_t          len;
 87         uint_t          prot;
 88         pfn_t           pfn;

 90         if (khat_running)
 91                 panic("va_to_pfn(): called too late\n");

 93         if (kbm_probe(&va, &len, &pfn, &prot) == 0)
 94                 return (PFN_INVALID);
 95         if (va > des_va)
 96                 return (PFN_INVALID);
 97         if (va < des_va)
 98                 pfn += mmu_btop(des_va - va);
 99         return (pfn);
100 }
```

unchanged_portion_omitted_

```
219 #endif

221 /*
222  * Routine to pre-allocate data structures for hat_kern_setup(). It computes
```



```
223  * how many pagetables it needs by walking the boot loader's page tables.
224  */
225 /*ARGSUSED*/
226 void
227 hat_kern_alloc(
228         caddr_t segmap_base,
229         size_t  segmap_size,
230         caddr_t ekernelheap)
231 {
232         uintptr_t       last_va = (uintptr_t)-1;        /* catch 1st time */
233         uintptr_t       va = 0;
234         size_t          size;
235         pfn_t           pfn;
236         uint_t          prot;
237         uint_t          table_cnt = 1;
238         uint_t          mapping_cnt;
239         level_t         start_level;
240         level_t         l;
241         struct memlist  *pmem;
242         level_t         lpagel = mmu.max_page_level;
243         uint64_t        paddr;
244         int64_t         psize;
245         int             nwindows;

247         if (kpm_size > 0) {
248                 /*
249                  * Create the kpm page tables.  When running on the
250                  * hypervisor these are made read/only at first.
251                  * Later we'll add write permission where possible.
252                  */
253                 for (pmem = phys_install; pmem; pmem = pmem->ml_next) {
254                         paddr = pmem->ml_address;
255                         psize = pmem->ml_size;
256                         while (psize >= MMU_PAGESIZE) {
257                                 /* find the largest page size */
258                                 for (l = lpagel; l > 0; l--) {
259                                         if ((paddr & LEVEL_OFFSET(l)) == 0 &&
260                                             psize > LEVEL_SIZE(l))
261                                                 break;
262                                 }

264 #if defined(__xpv)
265                                 /*
266                                  * Create read/only mappings to avoid
267                                  * conflicting with pagetable usage
268                                  */
269                                 xen_kpm_create(paddr, l);
270 #else
271                                 kbm_map((uintptr_t)kpm_vbase + paddr, paddr,
272                                     l, 1);
273 #endif
274                                 paddr += LEVEL_SIZE(l);
275                                 psize -= LEVEL_SIZE(l);
276                         }
277                 }
278         }

280         /*
281          * If this machine doesn't have a kpm segment, we need to allocate
282          * a small number of 'windows' which can be used to map pagetables.
283          */
284         nwindows = (kpm_size == 0) ? 2 * NCPU : 0;

286 #if defined(__xpv)
287         /*
288          * On a hypervisor, these windows are also used by the xpv_panic
```

```
289          * code, where we need one window for each level of the pagetable
290          * hierarchy.
291          */
292         nwindows = MAX(nwindows, mmu.max_level);
293 #endif

295         if (nwindows != 0) {
296                 /*
297                  * Create the page windows and 1 page of VA in
298                  * which we map the PTEs of those windows.
299                  */
300                 mmu.pwin_base = vmem_xalloc(heap_arena, nwindows * MMU_PAGESIZE,
301                     LEVEL_SIZE(1), 0, 0, NULL, NULL, VM_SLEEP);
302                 ASSERT(nwindows <= MMU_PAGESIZE / mmu.pte_size);
303                 mmu.pwin_pte_va = vmem_xalloc(heap_arena, MMU_PAGESIZE,
304                     MMU_PAGESIZE, 0, 0, NULL, NULL, VM_SLEEP);

306                 /*
307                  * Find/Create the page table window mappings.
308                  */
309                 paddr = 0;
310                 (void) find_pte((uintptr_t)mmu.pwin_base, &paddr, 0, 0);
311                 ASSERT(paddr != 0);
312                 ASSERT((paddr & MMU_PAGEOFFSET) == 0);
313                 mmu.pwin_pte_pa = paddr;
314 #ifdef __xpv
315                 (void) find_pte((uintptr_t)mmu.pwin_pte_va, NULL, 0, 0);
316                 kbm_read_only((uintptr_t)mmu.pwin_pte_va, mmu.pwin_pte_pa);
317 #else
318                 kbm_map((uintptr_t)mmu.pwin_pte_va, mmu.pwin_pte_pa, 0, 1);
319 #endif
320         }

322         /*
323          * Walk the boot loader's page tables and figure out
324          * how many tables and page mappings there will be.
325          */
326         while (kbm_probe(&va, &size, &pfn, &prot) != 0) {
327                 /*
328                  * At each level, if the last_va falls into a new htable,
329                  * increment table_cnt. We can stop at the 1st level where
330                  * they are in the same htable.
331                  */
332                 start_level = 0;
333                 while (start_level <= mmu.max_page_level) {
334                         if (size == LEVEL_SIZE(start_level))
335                                 break;
336                         start_level++;
337                 }

339                 for (l = start_level; l < mmu.max_level; ++l) {
340                         if (va >> LEVEL_SHIFT(l + 1) ==
341                             last_va >> LEVEL_SHIFT(l + 1))
342                                 break;
343                         ++table_cnt;
344                 }
345                 last_va = va;
346                 l = (start_level == 0) ? 1 : start_level;
347                 va = (va & LEVEL_MASK(l)) + LEVEL_SIZE(l);
348         }

350         /*
351          * Besides the boot loader mappings, we're going to fill in
352          * the entire top level page table for the kernel. Make sure there's
353          * enough reserve for that too.
354          */
```



```
355         table_cnt += mmu.top_level_count - ((kernelbase >>
356             LEVEL_SHIFT(mmu.max_level)) & (mmu.top_level_count - 1));

439 #if defined(__i386)
358         /*
441          * The 32 bit PAE hat allocates tables one level below the top when
442          * kernelbase isn't 1 Gig aligned. We'll just be sloppy and allocate
443          * a bunch more to the reserve. Any unused will be returned later.
444          * Note we've already counted these mappings, just not the extra
445          * pagetables.
446          */
447         if (mmu.pae_hat != 0 && (kernelbase & LEVEL_OFFSET(mmu.max_level)) != 0)
448                 table_cnt += mmu.ptes_per_table -
449                     ((kernelbase & LEVEL_OFFSET(mmu.max_level)) >>
450                     LEVEL_SHIFT(mmu.max_level - 1));
451 #endif

453         /*
359          * Add 1/4 more into table_cnt for extra slop.  The unused
360          * slop is freed back when we htable_adjust_reserve() later.
361          */
362         table_cnt += table_cnt >> 2;

364         /*
365          * We only need mapping entries (hments) for shared pages.
366          * This should be far, far fewer than the total possible,
367          * We'll allocate enough for 1/16 of all possible PTEs.
368          */
369         mapping_cnt = (table_cnt * mmu.ptes_per_table) >> 4;

371         /*
372          * Now create the initial htable/hment reserves
373          */
374         htable_initial_reserve(table_cnt);
375         hment_reserve(mapping_cnt);
376         x86pte_cpu_init(CPU);
377 }


380 /*
381  * This routine handles the work of creating the kernel's initial mappings
382  * by deciphering the mappings in the page tables created by the boot program.
383  *
384  * We maintain large page mappings, but only to a level 1 pagesize.
385  * The boot loader can only add new mappings once this function starts.
386  * In particular it can not change the pagesize used for any existing
387  * mappings or this code breaks!
388  */

390 void
391 hat_kern_setup(void)
392 {
393         /*
394          * Attach htables to the existing pagetables
395          */
396         /* BEGIN CSTYLED */
397         htable_attach(kas.a_hat, 0, mmu.max_level, NULL,
398 #ifdef __xpv
399             mmu_btop(xen_info->pt_base - ONE_GIG));
400 #else
401             mmu_btop(getcr3_pa()));
496             mmu_btop(getcr3()));
402 #endif
403         /* END CSTYLED */

405 #if defined(__xpv)
```

```
#if defined(__i386) && !defined(__xpv)
        CPU->cpu_tss->tss_cr3 = dftss0->tss_cr3 = getcr3();
#endif /* __i386 */

#if defined(__xpv) && defined(__amd64)
        /*
         * Try to make the kpm mappings r/w. Failures here are OK, as
         * it's probably just a pagetable
         */
        xen_kpm_finish_init();
#endif

        /*
         * The kernel HAT is now officially open for business.
         */
        khat_running = 1;

        CPUSET_ATOMIC_ADD(kas.a_hat->hat_cpus, CPU->cpu_id);
        CPU->cpu_current_hat = kas.a_hat;
}

#ifndef __xpv

/*
 * Note that the INVPCID_ALL* variants can be used even in the !PCIDE case, but
 * INVPCID_ADDR isn't.
 */
static void
invpcid(uint64_t type, uint64_t pcid, uintptr_t addr)
{
        ulong_t flag;
        uint64_t cr4;

        if (x86_use_invpcid == 1) {
                ASSERT(is_x86_feature(x86_featureset, X86FSET_INVPCID));
                invpcid_insn(type, pcid, addr);
                return;
        }

        switch (type) {
        case INVPCID_ALL_GLOBAL:
                flag = intr_clear();
                cr4 = getcr4();
                setcr4(cr4 & ~(ulong_t)CR4_PGE);
                setcr4(cr4 | CR4_PGE);
                intr_restore(flag);
                break;

        case INVPCID_ALL_NONGLOBAL:
                if (!(getcr4() & CR4_PCIDE)) {
                        reload_cr3();
                } else {
                        flag = intr_clear();
                        cr4 = getcr4();
                        setcr4(cr4 & ~(ulong_t)CR4_PGE);
                        setcr4(cr4 | CR4_PGE);
                        intr_restore(flag);
                }
                break;

        case INVPCID_ADDR:
                if (pcid == PCID_USER) {
                        flag = intr_clear();
                        ASSERT(addr < kernelbase);
                        ASSERT(ON_USER_HAT(CPU));
                        ASSERT(CPU->cpu_m.mcpu_kpti.kf_user_cr3 != 0);
```



```
                        tr_mmu_flush_user_range(addr, MMU_PAGESIZE,
                            MMU_PAGESIZE, CPU->cpu_m.mcpu_kpti.kf_user_cr3);
                        intr_restore(flag);
                } else {
                        mmu_invlpg((caddr_t)addr);
                }
                break;

        default:
                panic("unsupported invpcid(%lu)", type);
                break;
        }
}

/*
 * Flush one kernel mapping.
 *
 * We want to assert on kernel space here mainly for reasoning about the PCIDE
 * case: namely, this flush should never need to flush a non-current PCID
 * mapping.  This presumes we never have reason to flush the kernel regions
 * available to PCID_USER (the trampolines and so on).  It also relies on
 * PCID_KERNEL == PCID_NONE.
 */
void
mmu_flush_tlb_kpage(uintptr_t va)
{
        ASSERT(va >= kernelbase);
        ASSERT(getpcid() == PCID_KERNEL);
        mmu_invlpg((caddr_t)va);
}

/*
 * Flush one mapping: local CPU version of hat_tlb_inval().
 *
 * If this is a userspace address in the PCIDE case, we need two invalidations,
 * one for any potentially stale PCID_USER mapping, as well as any established
 * while in the kernel.
 */
void
mmu_flush_tlb_page(uintptr_t va)
{
        ASSERT(getpcid() == PCID_KERNEL);

        if (va >= kernelbase) {
                mmu_flush_tlb_kpage(va);
                return;
        }

        if (!(getcr4() & CR4_PCIDE)) {
                mmu_invlpg((caddr_t)va);
                return;
        }

        /*
         * Yes, kas will need to flush below kernelspace, at least during boot.
         * But there's no PCID_USER context.
         */
        if (ON_USER_HAT(CPU))
                invpcid(INVPCID_ADDR, PCID_USER, va);
        invpcid(INVPCID_ADDR, PCID_KERNEL, va);
}

static void
mmu_flush_tlb_range(uintptr_t addr, size_t len, size_t pgsz)
{
        EQUIV(addr < kernelbase, (addr + len - 1) < kernelbase);
```

```
533         ASSERT(len > 0);
534         ASSERT(pgsz != 0);

536         if (!(getcr4() & CR4_PCIDE) || x86_use_invpcid == 1) {
537                 for (uintptr_t va = addr; va < (addr + len); va += pgsz)
538                         mmu_flush_tlb_page(va);
539                 return;
540         }

542         /*
543          * As an emulated invpcid() in the PCIDE case requires jumping
544          * cr3s, we batch the invalidations.  We should only need to flush the
545          * user range if we're on a user-space HAT.
546          */
547         if (addr < kernelbase && ON_USER_HAT(CPU)) {
548                 ulong_t flag = intr_clear();
549                 ASSERT(CPU->cpu_m.mcpu_kpti.kf_user_cr3 != 0);
550                 tr_mmu_flush_user_range(addr, len, pgsz,
551                     CPU->cpu_m.mcpu_kpti.kf_user_cr3);
552                 intr_restore(flag);
553         }

555         for (uintptr_t va = addr; va < (addr + len); va += pgsz)
556                 mmu_invlpg((caddr_t)va);
557 }

559 /*
560  * MMU TLB (and PT cache) flushing on this CPU.
561  *
562  * FLUSH_TLB_ALL: invalidate everything, all PCIDs, all PT_GLOBAL.
563  * FLUSH_TLB_NONGLOBAL: invalidate all PCIDs, excluding PT_GLOBAL
564  * FLUSH_TLB_RANGE: invalidate the given range, including PCID_USER
565  * mappings as appropriate.  If using invpcid, PT_GLOBAL mappings are not
566  * invalidated.
567  */
568 void
569 mmu_flush_tlb(flush_tlb_type_t type, tlb_range_t *range)
570 {
571         ASSERT(getpcid() == PCID_KERNEL);

573         switch (type) {
574         case FLUSH_TLB_ALL:
575                 ASSERT(range == NULL);
576                 invpcid(INVPCID_ALL_GLOBAL, 0, 0);
577                 break;

579         case FLUSH_TLB_NONGLOBAL:
580                 ASSERT(range == NULL);
581                 invpcid(INVPCID_ALL_NONGLOBAL, 0, 0);
582                 break;

584         case FLUSH_TLB_RANGE: {
585                 mmu_flush_tlb_range(range->tr_va, TLB_RANGE_LEN(range),
586                     LEVEL_SIZE(range->tr_level));
587                 break;
588         }

590         default:
591                 panic("invalid call mmu_flush_tlb(%d)", type);
592                 break;
593         }
594 }

596 #endif /* ! __xpv */
```

```
**********************************************************
   11762 Fri Apr  6 17:25:06 2018
new/usr/src/uts/i86pc/vm/kboot_mmu.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

#include <sys/types.h>
#include <sys/systm.h>
#include <sys/archsystm.h>
#include <sys/debug.h>
#include <sys/bootconf.h>
#include <sys/bootsvcs.h>
#include <sys/bootinfo.h>
#include <sys/mman.h>
#include <sys/cmn_err.h>
#include <sys/param.h>
#include <sys/machparam.h>
#include <sys/machsystm.h>
#include <sys/promif.h>
#include <sys/kobj.h>
#ifdef __xpv
#include <sys/hypervisor.h>
#endif
#include <vm/kboot_mmu.h>
#include <vm/hat_pte.h>
#include <vm/hat_i86.h>
#include <vm/seg_kmem.h>

#if 0
/*
 * Joe's debug printing
 */
#define DBG(x)    \
        bop_printf(NULL, "kboot_mmu.c: %s is %" PRIx64 "\n", #x, (uint64_t)(x));
#else
#define DBG(x)  /* naught */
#endif
```



```
/*
 * Page table and memory stuff.
 */
static caddr_t window;
static caddr_t pte_to_window;

/*
 * this are needed by mmu_init()
 */
int kbm_nx_support = 0;         /* NX bit in PTEs is in use */
int kbm_pae_support = 0;        /* PAE is 64 bit Page table entries */
int kbm_pge_support = 0;        /* PGE is Page table global bit enabled */
int kbm_largepage_support = 0;
uint_t kbm_nucleus_size = 0;

#define BOOT_SHIFT(l)   (shift_amt[l])
#define BOOT_SZ(l)      ((size_t)1 << BOOT_SHIFT(l))
#define BOOT_OFFSET(l)  (BOOT_SZ(l) - 1)
#define BOOT_MASK(l)    (~BOOT_OFFSET(l))

/*
 * Initialize memory management parameters for boot time page table management
 */
void
kbm_init(struct xboot_info *bi)
{
        /*
         * configure mmu information
         */
        kbm_nucleus_size = (uintptr_t)bi->bi_kseg_size;
        kbm_largepage_support = bi->bi_use_largepage;
        kbm_nx_support = bi->bi_use_nx;
        kbm_pae_support = bi->bi_use_pae;
        kbm_pge_support = bi->bi_use_pge;
        window = bi->bi_pt_window;
        DBG(window);
        pte_to_window = bi->bi_pte_to_pt_window;
        DBG(pte_to_window);
        if (kbm_pae_support) {
                shift_amt = shift_amt_pae;
                ptes_per_table = 512;
                pte_size = 8;
                lpagesize = TWO_MEG;
#ifdef __amd64
                top_level = 3;
#else
                top_level = 2;
#endif
        } else {
                shift_amt = shift_amt_nopae;
                ptes_per_table = 1024;
                pte_size = 4;
                lpagesize = FOUR_MEG;
                top_level = 1;
        }

#ifdef __xpv
        xen_info = bi->bi_xen_start_info;
        mfn_list = (mfn_t *)xen_info->mfn_list;
        DBG(mfn_list);
        mfn_count = xen_info->nr_pages;
        DBG(mfn_count);
#endif
        top_page_table = bi->bi_top_page_table;
        DBG(top_page_table);
```

```
126 }

128 /*
129  * Change the addressible page table window to point at a given page
130  */
131 /*ARGSUSED*/
132 void *
133 kbm_remap_window(paddr_t physaddr, int writeable)
134 {
135         x86pte_t pt_bits = PT_NOCONSIST | PT_VALID | PT_WRITABLE;

137         DBG(physaddr);

139 #ifdef __xpv
140         if (!writeable)
141                 pt_bits &= ~PT_WRITABLE;
142         if (HYPERVISOR_update_va_mapping((uintptr_t)window,
143             pa_to_ma(physaddr) | pt_bits, UVMF_INVLPG | UVMF_LOCAL) < 0)
144                 bop_panic("HYPERVISOR_update_va_mapping() failed");
145 #else
146         if (kbm_pae_support)
147                 *((x86pte_t *)pte_to_window) = physaddr | pt_bits;
148         else
149                 *((x86pte32_t *)pte_to_window) = physaddr | pt_bits;
150         mmu_invlpg(window);
148         mmu_tlbflush_entry(window);
151 #endif
152         DBG(window);
153         return (window);
154 }

156 /*
157  * Add a mapping for the physical page at the given virtual address.
158  */
159 void
160 kbm_map(uintptr_t va, paddr_t pa, uint_t level, uint_t is_kernel)
161 {
162         x86pte_t *ptep;
163         paddr_t pte_physaddr;
164         x86pte_t pteval;

166         if (khat_running)
167                 panic("kbm_map() called too late");

169         pteval = pa_to_ma(pa) | PT_NOCONSIST | PT_VALID | PT_WRITABLE;
170         if (level >= 1)
171                 pteval |= PT_PAGESIZE;
172         if (kbm_pge_support && is_kernel)
173                 pteval |= PT_GLOBAL;

175 #ifdef __xpv
176         /*
177          * try update_va_mapping first - fails if page table is missing.
178          */
179         if (HYPERVISOR_update_va_mapping(va, pteval,
180             UVMF_INVLPG | UVMF_LOCAL) == 0)
181                 return;
182 #endif

184         /*
185          * Find the pte that will map this address. This creates any
186          * missing intermediate level page tables.
187          */
188         ptep = find_pte(va, &pte_physaddr, level, 0);
189         if (ptep == NULL)
190                 bop_panic("kbm_map: find_pte returned NULL");
```



```
192 #ifdef __xpv
193         if (HYPERVISOR_update_va_mapping(va, pteval, UVMF_INVLPG | UVMF_LOCAL))
194                 bop_panic("HYPERVISOR_update_va_mapping() failed");
195 #else
196         if (kbm_pae_support)
197                 *ptep = pteval;
198         else
199                 *((x86pte32_t *)ptep) = pteval;
200         mmu_invlpg((caddr_t)va);
198         mmu_tlbflush_entry((caddr_t)va);
201 #endif
202 }
_____unchanged_portion_omitted_


329 /*
330  * Destroy a boot loader page table 4K mapping.
331  */
332 void
333 kbm_unmap(uintptr_t va)
334 {
335         if (khat_running)
336                 panic("kbm_unmap() called too late");
337         else {
338 #ifdef __xpv
339                 (void) HYPERVISOR_update_va_mapping(va, 0,
340                     UVMF_INVLPG | UVMF_LOCAL);
341 #else
342                 x86pte_t *ptep;
343                 level_t level = 0;
344                 uint_t  probe_only = 1;

346                 ptep = find_pte(va, NULL, level, probe_only);
347                 if (ptep == NULL)
348                         return;

350                 if (kbm_pae_support)
351                         *ptep = 0;
352                 else
353                         *((x86pte32_t *)ptep) = 0;
354                 mmu_invlpg((caddr_t)va);
352                 mmu_tlbflush_entry((caddr_t)va);
355 #endif
356         }
357 }


360 /*
361  * Change a boot loader page table 4K mapping.
362  * Returns the pfn of the old mapping.
363  */
364 pfn_t
365 kbm_remap(uintptr_t va, pfn_t pfn)
366 {
367         x86pte_t *ptep;
368         level_t level = 0;
369         uint_t  probe_only = 1;
370         x86pte_t pte_val = pa_to_ma(pfn_to_pa(pfn)) | PT_WRITABLE |
371             PT_NOCONSIST | PT_VALID;
372         x86pte_t old_pte;

374         if (khat_running)
375                 panic("kbm_remap() called too late");
376         ptep = find_pte(va, NULL, level, probe_only);
377         if (ptep == NULL)
```

```
378                 bop_panic("kbm_remap: find_pte returned NULL");

380         if (kbm_pae_support)
381                 old_pte = *ptep;
382         else
383                 old_pte = *((x86pte32_t *)ptep);

385 #ifdef __xpv
386         if (HYPERVISOR_update_va_mapping(va, pte_val, UVMF_INVLPG | UVMF_LOCAL))
387                 bop_panic("HYPERVISOR_update_va_mapping() failed");
388 #else
389         if (kbm_pae_support)
390                 *((x86pte_t *)ptep) = pte_val;
391         else
392                 *((x86pte32_t *)ptep) = pte_val;
393         mmu_invlpg((caddr_t)va);
391         mmu_tlbflush_entry((caddr_t)va);
394 #endif

396         if (!(old_pte & PT_VALID) || ma_to_pa(old_pte) == -1)
397                 return (PFN_INVALID);
398         return (mmu_btop(ma_to_pa(old_pte)));
399 }


402 /*
403  * Change a boot loader page table 4K mapping to read only.
404  */
405 void
406 kbm_read_only(uintptr_t va, paddr_t pa)
407 {
408         x86pte_t pte_val = pa_to_ma(pa) |
409             PT_NOCONSIST | PT_REF | PT_MOD | PT_VALID;

411 #ifdef __xpv
412         if (HYPERVISOR_update_va_mapping(va, pte_val, UVMF_INVLPG | UVMF_LOCAL))
413                 bop_panic("HYPERVISOR_update_va_mapping() failed");
414 #else
415         x86pte_t *ptep;
416         level_t level = 0;

418         ptep = find_pte(va, NULL, level, 0);
419         if (ptep == NULL)
420                 bop_panic("kbm_read_only: find_pte returned NULL");

422         if (kbm_pae_support)
423                 *ptep = pte_val;
424         else
425                 *((x86pte32_t *)ptep) = pte_val;
426         mmu_invlpg((caddr_t)va);
424         mmu_tlbflush_entry((caddr_t)va);
427 #endif
428 }
```

*unchanged_portion_omitted_*

```
452 void
453 kbm_pop(void)
454 {
455 #ifdef __xpv
456         if (HYPERVISOR_update_va_mapping((uintptr_t)window, save_pte,
457             UVMF_INVLPG | UVMF_LOCAL) < 0)
458                 bop_panic("HYPERVISOR_update_va_mapping() failed");
459 #else
460         if (kbm_pae_support)
461                 *((x86pte_t *)pte_to_window) = save_pte;
462         else
```



```
463                 *((x86pte32_t *)pte_to_window) = save_pte;
464         mmu_invlpg(window);
462         mmu_tlbflush_entry(window);
465 #endif
466 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   99871 Fri Apr  6 17:25:06 2018
new/usr/src/uts/i86pc/vm/vm_machdep.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 * Copyright 2018 Joyent, Inc.
 * Copyright 2016 Joyent, Inc.
 */

/* Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
/*       All Rights Reserved   */

/*
 * Portions of this source code were derived from Berkeley 4.3 BSD
 * under license from the Regents of the University of California.
 */

/*
 * UNIX machine dependent virtual memory support.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/systm.h>
#include <sys/user.h>
#include <sys/proc.h>
#include <sys/kmem.h>
#include <sys/vmem.h>
#include <sys/buf.h>
#include <sys/cpuvar.h>
#include <sys/lgrp.h>
#include <sys/disp.h>
#include <sys/vm.h>
#include <sys/mman.h>
#include <sys/vnode.h>
#include <sys/cred.h>
#include <sys/exec.h>
#include <sys/exechdr.h>
```



```
#include <sys/debug.h>
#include <sys/vmsystm.h>
#include <sys/swap.h>
#include <sys/dumphdr.h>
#include <sys/random.h>

#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_kp.h>
#include <vm/seg_vn.h>
#include <vm/page.h>
#include <vm/seg_kmem.h>
#include <vm/seg_kpm.h>
#include <vm/vm_dep.h>

#include <sys/cpu.h>
#include <sys/vm_machparam.h>
#include <sys/memlist.h>
#include <sys/bootconf.h> /* XXX the memlist stuff belongs in memlist_plat.h */
#include <vm/hat_i86.h>
#include <sys/x86_archext.h>
#include <sys/elf_386.h>
#include <sys/cmn_err.h>
#include <sys/archsystm.h>
#include <sys/machsystm.h>
#include <sys/secflags.h>

#include <sys/vtrace.h>
#include <sys/ddidmareq.h>
#include <sys/promif.h>
#include <sys/memnode.h>
#include <sys/stack.h>
#include <util/qsort.h>
#include <sys/taskq.h>

#ifdef __xpv

#include <sys/hypervisor.h>
#include <sys/xen_mmu.h>
#include <sys/balloon_impl.h>

/*
 * domain 0 pages usable for DMA are kept pre-allocated and kept in
 * distinct lists, ordered by increasing mfn.
 */
static kmutex_t io_pool_lock;
static kmutex_t contig_list_lock;
static page_t *io_pool_4g;      /* pool for 32 bit dma limited devices */
static page_t *io_pool_16m;     /* pool for 24 bit dma limited legacy devices */
static long io_pool_cnt;
static long io_pool_cnt_max = 0;
#define DEFAULT_IO_POOL_MIN     128
static long io_pool_cnt_min = DEFAULT_IO_POOL_MIN;
static long io_pool_cnt_lowater = 0;
static long io_pool_shrink_attempts; /* how many times did we try to shrink */
static long io_pool_shrinks;    /* how many times did we really shrink */
static long io_pool_grows;      /* how many times did we grow */
static mfn_t start_mfn = 1;
static caddr_t io_pool_kva;     /* use to alloc pages when needed */

static int create_contig_pfnlist(uint_t);

/*
 * percentage of phys mem to hold in the i/o pool
 */
```

```
125 #define DEFAULT_IO_POOL_PCT     2
126 static long io_pool_physmem_pct = DEFAULT_IO_POOL_PCT;
127 static void page_io_pool_sub(page_t **, page_t *, page_t *);
128 int ioalloc_dbg = 0;

130 #endif /* __xpv */

132 uint_t vac_colors = 1;

134 int largepagesupport = 0;
135 extern uint_t page_create_new;
136 extern uint_t page_create_exists;
137 extern uint_t page_create_putbacks;
138 /*
139  * Allow users to disable the kernel's use of SSE.
140  */
141 extern int use_sse_pagecopy, use_sse_pagezero;

143 /*
144  * combined memory ranges from mnode and memranges[] to manage single
145  * mnode/mtype dimension in the page lists.
146  */
147 typedef struct {
148         pfn_t   mnr_pfnlo;
149         pfn_t   mnr_pfnhi;
150         int     mnr_mnode;
151         int     mnr_memrange;           /* index into memranges[] */
152         int     mnr_next;               /* next lower PA mnoderange */
153         int     mnr_exists;
154         /* maintain page list stats */
155         pgcnt_t mnr_mt_clpgcnt;         /* cache list cnt */
156         pgcnt_t mnr_mt_flpgcnt[MMU_PAGE_SIZES]; /* free list cnt per szc */
157         pgcnt_t mnr_mt_totcnt;          /* sum of cache and free lists */
158 #ifdef DEBUG
159         struct mnr_mts {                /* mnode/mtype szc stats */
160                 pgcnt_t mnr_mts_pgcnt;
161                 int     mnr_mts_colors;
162                 pgcnt_t *mnr_mtsc_pgcnt;
163         }       *mnr_mts;
164 #endif
165 } mnoderange_t;
```

unchanged_portion_omitted_

```
290 #endif

292 uint_t mmu_page_sizes;

294 /* How many page sizes the users can see */
295 uint_t mmu_exported_page_sizes;

297 /* page sizes that legacy applications can see */
298 uint_t mmu_legacy_page_sizes;

300 /*
301  * Number of pages in 1 GB.  Don't enable automatic large pages if we have
302  * fewer than this many pages.
303  */
304 pgcnt_t shm_lpg_min_physmem = 1 << (30 - MMU_PAGESHIFT);
305 pgcnt_t privm_lpg_min_physmem = 1 << (30 - MMU_PAGESHIFT);

307 /*
308  * Maximum and default segment size tunables for user private
309  * and shared anon memory, and user text and initialized data.
310  * These can be patched via /etc/system to allow large pages
311  * to be used for mapping application private and shared anon memory.
312  */
313 size_t mcntl0_lpsize = MMU_PAGESIZE;
```



```
314 size_t max_uheap_lpsize = MMU_PAGESIZE;
315 size_t default_uheap_lpsize = MMU_PAGESIZE;
316 size_t max_ustack_lpsize = MMU_PAGESIZE;
317 size_t default_ustack_lpsize = MMU_PAGESIZE;
318 size_t max_privmap_lpsize = MMU_PAGESIZE;
319 size_t max_uidata_lpsize = MMU_PAGESIZE;
320 size_t max_utext_lpsize = MMU_PAGESIZE;
321 size_t max_shm_lpsize = MMU_PAGESIZE;


324 /*
325  * initialized by page_coloring_init().
326  */
327 uint_t  page_colors;
328 uint_t  page_colors_mask;
329 uint_t  page_coloring_shift;
330 int     cpu_page_colors;
331 static uint_t   l2_colors;

333 /*
334  * Page freelists and cachelists are dynamically allocated once mnoderangecnt
335  * and page_colors are calculated from the l2 cache n-way set size.  Within a
336  * mnode range, the page freelist and cachelist are hashed into bins based on
337  * color. This makes it easier to search for a page within a specific memory
338  * range.
339  */
340 #define PAGE_COLORS_MIN 16

342 page_t ****page_freelists;
343 page_t ***page_cachelists;


346 /*
347  * Used by page layer to know about page sizes
348  */
349 hw_pagesize_t hw_page_array[MAX_NUM_LEVEL + 1];

351 kmutex_t        *fpc_mutex[NPC_MUTEX];
352 kmutex_t        *cpc_mutex[NPC_MUTEX];

354 /* Lock to protect mnoderanges array for memory DR operations. */
355 static kmutex_t mnoderange_lock;

357 /*
358  * Only let one thread at a time try to coalesce large pages, to
359  * prevent them from working against each other.
360  */
361 static kmutex_t contig_lock;
362 #define CONTIG_LOCK()   mutex_enter(&contig_lock);
363 #define CONTIG_UNLOCK() mutex_exit(&contig_lock);

365 #define PFN_16M         (mmu_btop((uint64_t)0x1000000))

367 caddr_t
368 i86devmap(pfn_t pf, pgcnt_t pgcnt, uint_t prot)
369 {
370         caddr_t addr;
371         caddr_t addr1;
372         page_t *pp;

374         addr1 = addr = vmem_alloc(heap_arena, mmu_ptob(pgcnt), VM_SLEEP);

376         for (; pgcnt != 0; addr += MMU_PAGESIZE, ++pf, --pgcnt) {
377                 pp = page_numtopp_nolock(pf);
378                 if (pp == NULL) {
379                         hat_devload(kas.a_hat, addr, MMU_PAGESIZE, pf,
```

```
380                             prot | HAT_NOSYNC, HAT_LOAD_LOCK);
381                 } else {
382                         hat_memload(kas.a_hat, addr, pp,
383                             prot | HAT_NOSYNC, HAT_LOAD_LOCK);
384                 }
385         }

387         return (addr1);
388 }

390 /*
391  * This routine is like page_numtopp, but accepts only free pages, which
392  * it allocates (unfrees) and returns with the exclusive lock held.
393  * It is used by machdep.c/dma_init() to find contiguous free pages.
394  */
395 page_t *
396 page_numtopp_alloc(pfn_t pfnum)
397 {
398         page_t *pp;

400 retry:
401         pp = page_numtopp_nolock(pfnum);
402         if (pp == NULL) {
403                 return (NULL);
404         }

406         if (!page_trylock(pp, SE_EXCL)) {
407                 return (NULL);
408         }

410         if (page_pptonum(pp) != pfnum) {
411                 page_unlock(pp);
412                 goto retry;
413         }

415         if (!PP_ISFREE(pp)) {
416                 page_unlock(pp);
417                 return (NULL);
418         }
419         if (pp->p_szc) {
420                 page_demote_free_pages(pp);
421                 page_unlock(pp);
422                 goto retry;
423         }

425         /* If associated with a vnode, destroy mappings */

427         if (pp->p_vnode) {

429                 page_destroy_free(pp);

431                 if (!page_lock(pp, SE_EXCL, (kmutex_t *)NULL, P_NO_RECLAIM)) {
432                         return (NULL);
433                 }

435                 if (page_pptonum(pp) != pfnum) {
436                         page_unlock(pp);
437                         goto retry;
438                 }
439         }

441         if (!PP_ISFREE(pp)) {
442                 page_unlock(pp);
443                 return (NULL);
444         }
```



```
446         if (!page_reclaim(pp, (kmutex_t *)NULL))
447                 return (NULL);

449         return (pp);
450 }

452 /*
453  * Return the optimum page size for a given mapping
454  */
455 /*ARGSUSED*/
456 size_t
457 map_pgsz(int maptype, struct proc *p, caddr_t addr, size_t len, int memcntl)
458 {
459         level_t l = 0;
460         size_t pgsz = MMU_PAGESIZE;
461         size_t max_lpsize;
462         uint_t mszc;

464         ASSERT(maptype != MAPPGSZ_VA);

466         if (maptype != MAPPGSZ_ISM && physmem < privm_lpg_min_physmem) {
467                 return (MMU_PAGESIZE);
468         }

470         switch (maptype) {
471         case MAPPGSZ_HEAP:
472         case MAPPGSZ_STK:
473                 max_lpsize = memcntl ? mcntl0_lpsize : (maptype ==
474                     MAPPGSZ_HEAP ? max_uheap_lpsize : max_ustack_lpsize);
475                 if (max_lpsize == MMU_PAGESIZE) {
476                         return (MMU_PAGESIZE);
477                 }
478                 if (len == 0) {
479                         len = (maptype == MAPPGSZ_HEAP) ? p->p_brkbase +
480                             p->p_brksize - p->p_bssbase : p->p_stksize;
481                 }
482                 len = (maptype == MAPPGSZ_HEAP) ? MAX(len,
483                     default_uheap_lpsize) : MAX(len, default_ustack_lpsize);

485                 /*
486                  * use the pages size that best fits len
487                  */
488                 for (l = mmu.umax_page_level; l > 0; --l) {
489                         if (LEVEL_SIZE(l) > max_lpsize || len < LEVEL_SIZE(l)) {
490                                 continue;
491                         } else {
492                                 pgsz = LEVEL_SIZE(l);
493                         }
494                         break;
495                 }

497                 mszc = (maptype == MAPPGSZ_HEAP ? p->p_brkpageszc :
498                     p->p_stkpageszc);
499                 if (addr == 0 && (pgsz < hw_page_array[mszc].hp_size)) {
500                         pgsz = hw_page_array[mszc].hp_size;
501                 }
502                 return (pgsz);

504         case MAPPGSZ_ISM:
505                 for (l = mmu.umax_page_level; l > 0; --l) {
506                         if (len >= LEVEL_SIZE(l))
507                                 return (LEVEL_SIZE(l));
508                 }
509                 return (LEVEL_SIZE(0));
510         }
511         return (pgsz);
```

```
 512 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5573 Fri Apr  6 17:25:07 2018
new/usr/src/uts/i86xpv/Makefile.files
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
#
# Copyright 2018 Joyent, Inc.
#

#       This Makefile defines file modules in the directory uts/i86xpv
#       and its children. These are the source files which are i86xpv
#       "implementation architecture" dependent.
#

#
#       object lists
#
CORE_OBJS +=                    \
        acpi_stubs.o            \
        balloon.o               \
        biosdisk.o              \
        cbe.o                   \
        cmi.o                   \
        cmi_hw.o                \
        cms.o                   \
        confunix.o              \
        cpuid.o                 \
        cpuid_subr.o            \
        cpupm.o                 \
        cpupm_mach.o            \
        dis_tables.o            \
        ddi_impl.o              \
        dtrace_subr.o           \
        dvma.o                  \
        fakebop.o               \
        fpu_subr.o              \
        fastboot.o              \
        fb_swtch.o              \
        graphics.o              \
        hardclk.o               \
        hat_i86.o               \
```



```
        hat_kdi.o               \
        hment.o                 \
        hold_page.o             \
        hrtimers.o              \
        htable.o                \
        i86_mmu.o               \
        ibft.o                  \
        instr_size.o            \
        intr.o                  \
        kboot_mmu.o             \
        kdi_subr.o              \
        kdi_idt.o               \
        kdi_idthdl.o            \
        kdi_asm.o               \
        lgrpplat.o              \
        mach_kdi.o              \
        mach_sysconfig.o        \
        machdep.o               \
        mem_config_stubs.o      \
        memnode.o               \
        microcode.o             \
        mlsetup.o               \
        mp_call.o               \
        mp_implfuncs.o          \
        mp_machdep.o            \
        mp_startup.o            \
        memscrub.o              \
        notes.o                 \
        pci_bios.o              \
        pci_cfgacc.o            \
        pci_cfgacc_x86.o        \
        pci_cfgspace.o          \
        pci_mech1.o             \
        pci_mech2.o             \
        pci_neptune.o           \
        pci_orion.o             \
        pmem.o                  \
        ppage.o                 \
        startup.o               \
        ssp.o                   \
        xpv_timestamp.o         \
        todpc_subr.o            \
        trap.o                  \
        vm_machdep.o            \
        x_call.o

#
#       Add the SMBIOS subsystem object files directly to the list of objects
#       built into unix itself; this is all common code except for smb_dev.c.
#
CORE_OBJS += $(SMBIOS_OBJS)

#
# These get compiled twice:
# - once in the dboot (direct boot) identity mapped code
# - once for use during early startup in unix
#
BOOT_DRIVER_OBJS =              \
        boot_console.o          \
        boot_keyboard.o         \
        boot_keyboard_table.o   \
        boot_mmu.o              \
        boot_vga.o              \
        boot_xconsole.o         \
        dboot_multiboot2.o      \
        $(FONT_OBJS)
```

```
126 CORE_OBJS += $(BOOT_DRIVER_OBJS)

128 #
129 # Extra XEN files separated out for now.
130 #
131 CORE_OBJS +=                    \
132         cpr_driver.o            \
133         evtchn.o                \
134         gnttab.o                \
135         hypercall.o             \
136         hyperevent.o            \
137         hypersubr.o             \
138         mp_xen.o                \
139         panic_asm.o             \
140         xenguest.o              \
141         xenbus_client.o         \
142         xenbus_comms.o          \
143         xenbus_probe.o          \
144         xenbus_xs.o             \
145         xen_machdep.o           \
146         xen_mmu.o               \
147         xpv_panic.o             \
148         xvdi.o

150 #
151 #       locore.o is special. It must be the first file relocated so that it
152 #       it is relocated just where its name implies.
153 #
154 SPECIAL_OBJS_32 +=              \
155         locore.o                \
156         fast_trap_asm.o         \
157         interrupt.o             \
158         syscall_asm.o

160 SPECIAL_OBJS_64 +=              \
161         locore.o                \
162         fast_trap_asm.o         \
163         interrupt.o             \
164         syscall_asm_amd64.o     \
165         kpti_trampolines.o
163         syscall_asm_amd64.o

167 SPECIAL_OBJS += $(SPECIAL_OBJS_$(CLASS))

169 #
170 # object files used to boot into full kernel
171 #
172 DBOOT_OBJS_32 = muldiv.o

174 DBOOT_OBJS_64 =

176 DBOOT_OBJS +=                   \
177         dboot_asm.o             \
178         dboot_printf.o          \
179         dboot_startkern.o       \
180         dboot_xen.o             \
181         hypercall.o             \
182         hypersubr.o             \
183         memcpy.o                \
184         memset.o                \
185         string.o                \
186         $(BOOT_DRIVER_OBJS)     \
187         $(DBOOT_OBJS_$(CLASS))

189 #
```



```
190 #                       driver & misc modules
191 #
192 BALLOON_OBJS += balloon_drv.o
193 DOMCAPS_OBJS += domcaps.o
194 EVTCHN_OBJS += evtchn_dev.o
195 GFX_PRIVATE_OBJS += gfx_private.o gfxp_pci.o gfxp_segmap.o \
196                     gfxp_devmap.o gfxp_vgatext.o gfxp_vm.o vgasubr.o
197 IOAT_OBJS += ioat.o ioat_rs.o ioat_ioctl.o ioat_chan.o
198 ISANEXUS_OBJS += isa.o dma_engine.o i8237A.o
199 PCI_E_NEXUS_OBJS += npe.o npe_misc.o
200 PCI_E_NEXUS_OBJS += pci_common.o pci_kstats.o pci_tools.o
201 PCINEXUS_OBJS += pci.o pci_common.o pci_kstats.o pci_tools.o
202 PRIVCMD_OBJS += seg_mf.o privcmd.o privcmd_hcall.o
203 ROOTNEX_OBJS += rootnex.o
204 XPVTOD_OBJS += xpvtod.o
205 XPV_AUTOCONFIG_OBJS += xpv_autoconfig.o
206 XPV_PSM_OBJS += xpv_psm.o mp_platform_common.o mp_platform_xpv.o \
207                 apic_regops.o psm_common.o xpv_intr.o
208 XPV_UPPC_OBJS += xpv_uppc.o psm_common.o
209 XENBUS_OBJS += xenbus_dev.o
210 XENCONS_OBJS += xencons.o
211 XPVD_OBJS += xpvd.o
212 XPVTAP_OBJS += xpvtap.o blk_common.o seg_mf.o
213 XNB_OBJS += xnb.o
214 XNBE_OBJS += xnbe.o
215 XNBO_OBJS += xnbo.o
216 XNBU_OBJS += xnbu.o
217 XNF_OBJS += xnf.o
218 XSVC_OBJS += xsvc.o
219 XDF_OBJS += xdf.o
220 XDB_OBJS += xdb.o
221 XDT_OBJS += xdt.o

223 #
224 #       Build up defines and paths.
225 #
226 INC_PATH        += -I$(UTSBASE)/i86xpv -I$(UTSBASE)/i86pc -I$(SRC)/common \
227         -I$(UTSBASE)/common/xen

229 #
230 # Since the assym files are derived, the dependencies must be explicit for
231 # all files including this file. (This is only actually required in the
232 # instance when the .nse_depinfo file does not exist.) It may seem that
233 # the lint targets should also have a similar dependency, but they don't
234 # since only C headers are included when #defined(__lint) is true.
235 #

237 ASSYM_DEPS      +=              \
238         copy.o                  \
239         desctbls_asm.o          \
240         ddi_i86_asm.o           \
241         exception.o             \
242         fast_trap_asm.o         \
243         float.o                 \
244         hyperevent.o            \
245         i86_subr.o              \
246         kdi_asm.o               \
247         interrupt.o             \
248         lock_prim.o             \
249         locore.o                \
250         panic_asm.o             \
251         sseblk.o                \
252         swtch.o                 \
253         syscall_asm.o           \
254         syscall_asm_amd64.o
```

```
256 $(KDI_ASSYM_DEPS:%=$(OBJS_DIR)/%):      $(DSF_DIR)/$(OBJS_DIR)/kdi_assym.h
256 ASSYM_DEPS += kdi_asm.o
```

```
**********************************************************
   27893 Fri Apr  6 17:25:07 2018
new/usr/src/uts/i86xpv/os/xpv_panic.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2012 Gary Mills
 * Copyright 2016 PALO, Richard.
 *
 * Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
 *
 * Copyright 2018 Joyent, Inc.
 */

#include <sys/types.h>
#include <sys/clock.h>
#include <sys/psm.h>
#include <sys/archsystm.h>
#include <sys/machsystm.h>
#include <sys/compress.h>
#include <sys/modctl.h>
#include <sys/trap.h>
#include <sys/panic.h>
#include <sys/regset.h>
#include <sys/frame.h>
#include <sys/kobj.h>
#include <sys/apic.h>
#include <sys/apic_timer.h>
#include <sys/dumphdr.h>
#include <sys/mem.h>
#include <sys/x86_archext.h>
#include <sys/xpv_panic.h>
#include <sys/boot_console.h>
#include <sys/bootsvcs.h>
#include <sys/consdev.h>
#include <vm/hat_pte.h>
#include <vm/hat_i86.h>

/* XXX: need to add a PAE version too, if we ever support both PAE and non */
#if defined(__i386)
#define XPV_FILENAME    "/boot/xen-syms"
#else
#define XPV_FILENAME    "/boot/amd64/xen-syms"
#endif
```



```
#define XPV_MODNAME     "xpv"

int xpv_panicking = 0;

struct module *xpv_module;
struct modctl *xpv_modctl;

#define ALIGN(x, a)     ((a) == 0 ? (uintptr_t)(x) : \
        (((uintptr_t)(x) + (uintptr_t)(a) - 1l) & ~((uintptr_t)(a) - 1l)))

/* Pointer to the xpv_panic_info structure handed to us by Xen.  */
static struct panic_info *xpv_panic_info = NULL;

/* Timer support */
#define NSEC_SHIFT 5
#define T_XPV_TIMER     0xd1
#define XPV_TIMER_INTERVAL      1000    /* 1000 microseconds */
static uint32_t *xpv_apicadr = NULL;
static uint_t   nsec_scale;

/* IDT support */
#pragma align   16(xpv_panic_idt)
static gate_desc_t      xpv_panic_idt[NIDT];    /* interrupt descriptor table */

/* Xen pagetables mapped into our HAT's ptable windows */
static pfn_t ptable_pfn[MAX_NUM_LEVEL];

/* Number of MMU_PAGESIZE pages we're adding to the Solaris dump */
static int xpv_dump_pages;

/*
 * There are up to two large swathes of RAM that we don't want to include
 * in the dump: those that comprise the Xen version of segkpm.  On 32-bit
 * systems there is no such region of memory.  On 64-bit systems, there
 * should be just a single contiguous region that corresponds to all of
 * physical memory.  The tricky bit is that Xen's heap sometimes lives in
 * the middle of their segkpm, and is mapped using only kpm-like addresses.
 * In that case, we need to skip the swathes before and after Xen's heap.
 */
uintptr_t kpm1_low = 0;
uintptr_t kpm1_high = 0;
uintptr_t kpm2_low = 0;
uintptr_t kpm2_high = 0;

/*
 * Some commonly used values that we don't want to recompute over and over.
 */
static int xpv_panic_nptes[MAX_NUM_LEVEL];
static ulong_t xpv_panic_cr3;
static uintptr_t xpv_end;

static void xpv_panic_console_print(const char *fmt, ...);
static void (*xpv_panic_printf)(const char *, ...) = xpv_panic_console_print;

#define CONSOLE_BUF_SIZE        256
static char console_buffer[CONSOLE_BUF_SIZE];
static boolean_t use_polledio;

/*
 * Pointers to machine check panic info (if any).
 */
xpv_mca_panic_data_t *xpv_mca_panic_data = NULL;

static void
xpv_panic_putc(int m)
{
```

```
126         struct cons_polledio *c = cons_polledio;

128         /* This really shouldn't happen */
129         if (boot_console_type(NULL) == CONS_HYPERVISOR)
130                 return;

132         if (use_polledio == B_TRUE)
133                 c->cons_polledio_putchar(c->cons_polledio_argument, m);
134         else
135                 bcons_putchar(m);
136 }
______unchanged_portion_omitted_

160 static void
161 xpv_panic_map(int level, pfn_t pfn)
162 {
163         x86pte_t pte, *pteptr;

165         /*
166          * The provided pfn represents a level 'level' page table.  Map it
167          * into the 'level' slot in the list of page table windows.
168          */
169         pteptr = (x86pte_t *)PWIN_PTE_VA(level);
170         pte = pfn_to_pa(pfn) | PT_VALID;

172         XPV_ALLOW_PAGETABLE_UPDATES();
173         if (mmu.pae_hat)
174                 *pteptr = pte;
175         else
176                 *(x86pte32_t *)pteptr = pte;
177         XPV_DISALLOW_PAGETABLE_UPDATES();

179         mmu_flush_tlb_page((uintptr_t)PWIN_VA(level));
177         mmu_tlbflush_entry(PWIN_VA(level));
180 }
______unchanged_portion_omitted_
```

```
**********************************************************
   13843 Fri Apr  6 17:25:08 2018
new/usr/src/uts/intel/Makefile.rules
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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.  All rights reserved.
# Copyright 2012 Joyent, Inc.  All rights reserved.
# Copyright 2017 Nexenta Systems, Inc.
#

#
#       This Makefile defines all file modules and build rules for the
# directory uts/intel and its children. These are the source files which
# are specific to the intel processor.
#
#       The following two-level ordering must be maintained in this file.
#         Lines are sorted first in order of decreasing specificity based on
#         the first directory component.  That is, sun4u rules come before
#         sparc rules come before common rules.
#
#         Lines whose initial directory components are equal are sorted
#         alphabetically by the remaining components.

#
# Need a way to distinguish between the ia32 and amd64 subdirs.
#
SUBARCH_DIR_32 = ia32
SUBARCH_DIR_64 = amd64
SUBARCH_DIR     = $(SUBARCH_DIR_$(CLASS))

#
#       Section 1a: C object build rules
#
$(OBJS_DIR)/%.o:                $(SRC)/common/fs/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/common/io/power/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)
```



```
$(OBJS_DIR)/%.o:                $(SRC)/common/util/i386/%.s
        $(COMPILE.s) -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/brand/sn1/%.s
        $(COMPILE.s) -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/brand/solaris10/%.s
        $(COMPILE.s) -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/dtrace/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/dtrace/%.s
        $(COMPILE.s) -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/fs/proc/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/ia32/ml/%.s
        $(COMPILE.s) -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/ia32/os/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/ia32/promif/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/ia32/syscall/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/%.s
        $(COMPILE.s) -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/debugger/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/events/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/hardware/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/dispatcher/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/executer/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)
```

```
$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/parser/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/namespace/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/resources/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/tables/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/utilities/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/acpica/disassembler/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/agpgart/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/agpmaster/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/amd8111s/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/amr/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/drm/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/hotplug/pcicfg/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/ipmi/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/intel_nb5000/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/intel_nhm/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(SRC)/common/mc/mc-amd/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/mc-amd/%.c
        $(COMPILE.c) -o $@ $<
```



```
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/pci/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/pciex/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/dktp/controller/ata/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/dktp/dcdev/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/dktp/disk/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/dktp/drvobj/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/dktp/hba/ghd/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/dnet/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/scsi/adapters/arcmsr/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/scsi/targets/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/vgatext/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/vmxnet3s/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/io/scsi/adapters/pvscsi/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/nskern/%.s
        $(COMPILE.s) -o $@ $<

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/os/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/pcbe/%.c
        $(COMPILE.c) -o $@ $<
        $(CTFCONVERT_O)

$(OBJS_DIR)/%.o:                $(UTSBASE)/intel/promif/%.c
```

```
257         $(COMPILE.c) -o $@ $<
258         $(CTFCONVERT_O)

260 $(OBJS_DIR)/%.o:                $(UTSBASE)/intel/syscall/%.c
261         $(COMPILE.c) -o $@ $<
262         $(CTFCONVERT_O)

264 $(OBJS_DIR)/%.o:                $(UTSBASE)/common/os/%.c
265         $(COMPILE.c) -o $@ $<
266         $(CTFCONVERT_O)

268 $(OBJS_DIR)/%.o:                $(UTSBASE)/intel/kdi/%.c
269         $(COMPILE.c) -o $@ $<
270         $(CTFCONVERT_O)

272 $(OBJS_DIR)/%.o:                $(UTSBASE)/intel/kdi/%.s
273         $(COMPILE.s) -o $@ $<

275 $(OBJS_DIR)/%.o:                $(UTSBASE)/intel/kdi/$(SUBARCH_DIR)/%.s
276         $(COMPILE.s) -o $@ $<

275 $(OBJS_DIR)/%.o:                $(UTSBASE)/intel/zfs/%.c
276         $(COMPILE.c) -o $@ $<
277         $(CTFCONVERT_O)

279 #
280 # krtld compiled into unix
281 #

283 KRTLD_INC_PATH  = -I$(UTSBASE)/common/krtld -I$(UTSBASE)/intel/sys
284 KRTLD_INC_PATH  += -I$(UTSBASE)/intel/$(SUBARCH_DIR)/krtld

286 KRTLD_CPPFLAGS_32       = -DELF_TARGET_386
287 KRTLD_CPPFLAGS_64       = -DELF_TARGET_AMD64 -DMODDIR_SUFFIX=\"amd64\"
288 KRTLD_CPPFLAGS          = $(KRTLD_CPPFLAGS_$(CLASS)) -D_KRTLD

290 $(OBJS_DIR)/%.o:        $(UTSBASE)/common/krtld/%.c
291         $(COMPILE.c) $(KRTLD_INC_PATH) $(KRTLD_CPPFLAGS) -o $@ $<
292         $(CTFCONVERT_O)

294 $(OBJS_DIR)/%.o:        $(UTSBASE)/intel/$(SUBARCH_DIR)/krtld/%.c
295         $(COMPILE.c) $(KRTLD_INC_PATH) $(KRTLD_CPPFLAGS) -o $@ $<
296         $(CTFCONVERT_O)

298 #
299 # _DBOOT indicates that krtld is called from a dboot ELF section
300 #
301 $(OBJS_DIR)/kobj.o              := CPPFLAGS += -D_DBOOT

303 $(OBJS_DIR)/%.o:        $(UTSBASE)/intel/$(SUBARCH_DIR)/krtld/%.s
304         $(COMPILE.s) $(KRTLD_INC_PATH) $(KRTLD_CPPFLAGS) -o $@ $<
305         $(CTFCONVERT_O)

307 $(OBJS_DIR)/%.o:        $(SRC)/common/util/$(SUBARCH_DIR)/%.c
308         $(COMPILE.c) $(KRTLD_INC_PATH) $(KRTLD_CPPFLAGS) -o $@ $<
309         $(CTFCONVERT_O)


312 #
313 #       Section 1b: Lint 'object' build rules.
314 #
315 $(LINTS_DIR)/%.ln:              $(SRC)/common/fs/%.c
316         @($(LHEAD) $(LINT.c) $< $(LTAIL))

318 $(LINTS_DIR)/%.ln:              $(SRC)/common/util/i386/%.s
319         @($(LHEAD) $(LINT.s) $< $(LTAIL))
```



```
321 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/brand/sn1/%.s
322         @($(LHEAD) $(LINT.s) $< $(LTAIL))

324 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/brand/solaris10/%.s
325         @($(LHEAD) $(LINT.s) $< $(LTAIL))

327 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/dtrace/%.c
328         @($(LHEAD) $(LINT.c) $< $(LTAIL))

330 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/dtrace/%.s
331         @($(LHEAD) $(LINT.s) $< $(LTAIL))

333 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/zfs/%.c
334         @($(LHEAD) $(LINT.c) $< $(LTAIL))

336 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/fs/proc/%.c
337         @($(LHEAD) $(LINT.c) $< $(LTAIL))

339 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/ia32/ml/%.s
340         @($(LHEAD) $(LINT.s) $< $(LTAIL))

342 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/ia32/os/%.c
343         @($(LHEAD) $(LINT.c) $< $(LTAIL))

345 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/ia32/promif/%.c
346         @($(LHEAD) $(LINT.c) $< $(LTAIL))

348 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/ia32/syscall/%.c
349         @($(LHEAD) $(LINT.c) $< $(LTAIL))

351 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/%.c
352         @($(LHEAD) $(LINT.c) $< $(LTAIL))

354 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/%.s
355         @($(LHEAD) $(LINT.s) $< $(LTAIL))

357 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/%.c
358         @($(LHEAD) $(LINT.c) $< $(LTAIL))

360 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/debugger/%.c
361         @($(LHEAD) $(LINT.c) $< $(LTAIL))

363 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/events/%.c
364         @($(LHEAD) $(LINT.c) $< $(LTAIL))

366 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/hardware/%.c
367         @($(LHEAD) $(LINT.c) $< $(LTAIL))

369 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/dispatcher/%.c
370         @($(LHEAD) $(LINT.c) $< $(LTAIL))

372 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/executer/%.c
373         @($(LHEAD) $(LINT.c) $< $(LTAIL))

375 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/parser/%.c
376         @($(LHEAD) $(LINT.c) $< $(LTAIL))

378 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/namespace/%.c
379         @($(LHEAD) $(LINT.c) $< $(LTAIL))

381 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/resources/%.c
382         @($(LHEAD) $(LINT.c) $< $(LTAIL))

384 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/tables/%.c
385         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```

```
387 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/utilities/%.c
388         @($(LHEAD) $(LINT.c) $< $(LTAIL))

390 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/acpica/disassembler/%.c
391         @($(LHEAD) $(LINT.c) $< $(LTAIL))

393 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/agpgart/%.c
394         @($(LHEAD) $(LINT.c) $< $(LTAIL))

396 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/agpmaster/%.c
397         @($(LHEAD) $(LINT.c) $< $(LTAIL))

399 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/amd8111s/%.c
400         @($(LHEAD) $(LINT.c) $< $(LTAIL))

402 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/amr/%.c
403         @($(LHEAD) $(LINT.c) $< $(LTAIL))

405 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/scsi/adapters/arcmsr/%.c
406         @($(LHEAD) $(LINT.c) $< $(LTAIL))

408 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/drm/%.c
409         @($(LHEAD) $(LINT.c) $< $(LTAIL))

411 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/hotplug/pcicfg/%.c
412         @($(LHEAD) $(LINT.c) $< $(LTAIL))

414 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/hotplug/pci/%.c
415         @($(LHEAD) $(LINT.c) $< $(LTAIL))

417 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/intel_nb5000/%.c
418         @($(LHEAD) $(LINT.c) $< $(LTAIL))

420 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/intel_nhm/%.c
421         @($(LHEAD) $(LINT.c) $< $(LTAIL))

423 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/ipmi/%.c
424         @($(LHEAD) $(LINT.c) $< $(LTAIL))

426 $(LINTS_DIR)/%.ln:              $(SRC)/common/mc/mc-amd/%.c
427         @($(LHEAD) $(LINT.c) $< $(LTAIL))

429 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/mc-amd/%.c
430         @($(LHEAD) $(LINT.c) $< $(LTAIL))

432 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/pci/%.c
433         @($(LHEAD) $(LINT.c) $< $(LTAIL))

435 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/pciex/%.c
436         @($(LHEAD) $(LINT.c) $< $(LTAIL))

438 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/dktp/controller/ata/%.c
439         @($(LHEAD) $(LINT.c) $< $(LTAIL))

441 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/dktp/dcdev/%.c
442         @($(LHEAD) $(LINT.c) $< $(LTAIL))

444 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/dktp/disk/%.c
445         @($(LHEAD) $(LINT.c) $< $(LTAIL))

447 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/dktp/drvobj/%.c
448         @($(LHEAD) $(LINT.c) $< $(LTAIL))

450 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/dktp/hba/ghd/%.c
451         @($(LHEAD) $(LINT.c) $< $(LTAIL))
```



```
453 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/dnet/%.c
454         @($(LHEAD) $(LINT.c) $< $(LTAIL))

456 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/scsi/targets/%.c
457         @($(LHEAD) $(LINT.c) $< $(LTAIL))

459 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/vgatext/%.c
460         @($(LHEAD) $(LINT.c) $< $(LTAIL))

462 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/vmxnet3s/%.c
463         @($(LHEAD) $(LINT.c) $< $(LTAIL))

465 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/io/scsi/adapters/pvscsi/%.c
466         @($(LHEAD) $(LINT.c) $< $(LTAIL))

468 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/os/%.c
469         @($(LHEAD) $(LINT.c) $< $(LTAIL))

471 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/pcbe/%.c
472         @($(LHEAD) $(LINT.c) $< $(LTAIL))

474 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/promif/%.c
475         @($(LHEAD) $(LINT.c) $< $(LTAIL))

477 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/syscall/%.c
478         @($(LHEAD) $(LINT.c) $< $(LTAIL))

480 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/os/%.c
481         @($(LHEAD) $(LINT.c) $< $(LTAIL))

483 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/kdi/%.c
484         @($(LHEAD) $(LINT.c) $< $(LTAIL))

486 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/kdi/%.s
487         @($(LHEAD) $(LINT.s) $< $(LTAIL))

492 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/kdi/$(SUBARCH_DIR)/%.s
493         @($(LHEAD) $(LINT.s) $< $(LTAIL))

489 $(LINTS_DIR)/%.ln:              $(UTSBASE)/intel/nskern/%.s
490         @($(LHEAD) $(LINT.s) $< $(LTAIL))

492 #
493 # krtld lints
494 #
495 $(LINTS_DIR)/%.ln:              $(UTSBASE)/common/krtld/%.c
496         @($(LHEAD) $(LINT.c) $(KRTLD_INC_PATH) $(KRTLD_CPPFLAGS) $< $(LTAIL))

498 $(LINTS_DIR)/%.ln:      $(UTSBASE)/intel/$(SUBARCH_DIR)/krtld/%.c
499         @($(LHEAD) $(LINT.c) $(KRTLD_INC_PATH) $(KRTLD_CPPFLAGS) $< $(LTAIL))

501 $(LINTS_DIR)/%.ln:      $(UTSBASE)/intel/$(SUBARCH_DIR)/krtld/%.s
502         @($(LHEAD) $(LINT.s) $(KRTLD_INC_PATH) $(KRTLD_CPPFLAGS) $< $(LTAIL))

504 $(LINTS_DIR)/%.ln:      $(SRC)/common/util/$(SUBARCH_DIR)/%.c
505         @($(LHEAD) $(LINT.c) $(KRTLD_INC_PATH) $(KRTLD_CPPFLAGS) $< $(LTAIL))

507 $(OBJS_DIR)/kobj.ln             := CPPFLAGS += -D_DBOOT
```

```
**********************************************************
    2035 Fri Apr  6 17:25:08 2018
new/usr/src/uts/intel/amd64/sys/kdi_regs.h
9441 kmdb should stash %cr3 in kdiregs
Reviewed by: John Levon <john.levon@joyent.com>
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
 1 /*
 2  * CDDL HEADER START
 3  *
 4  * The contents of this file are subject to the terms of the
 5  * Common Development and Distribution License (the "License").
 6  * You may not use this file except in compliance with the License.
 7  *
 8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 9  * or http://www.opensolaris.org/os/licensing.
10  * See the License for the specific language governing permissions
11  * and limitations under the License.
12  *
13  * When distributing Covered Code, include this CDDL HEADER in each
14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
15  * If applicable, add the following below this CDDL HEADER, with the
16  * fields enclosed by brackets "[]" replaced with your own identifying
17  * information: Portions Copyright [yyyy] [name of copyright owner]
18  *
19  * CDDL HEADER END
20  */

22 /*
23  * Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
24  * Use is subject to license terms.
25  *
26  * Copyright 2018 Joyent, Inc.
27  */

29 #ifndef _AMD64_SYS_KDI_REGS_H
30 #define _AMD64_SYS_KDI_REGS_H

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

32 #ifdef __cplusplus
33 extern "C" {
34 #endif

36 #define KDIREG_NGREG    31

36 /*
37  * A modified version of struct regs layout.
38  */

40 #define KDIREG_SAVFP    0
41 #define KDIREG_SAVPC    1
42 #define KDIREG_RDI      2
43 #define KDIREG_RSI      3
44 #define KDIREG_RDX      4
45 #define KDIREG_RCX      5
46 #define KDIREG_R8       6
47 #define KDIREG_R9       7
48 #define KDIREG_RAX      8
```



```
49 #define KDIREG_RBX      9
50 #define KDIREG_RBP      10
51 #define KDIREG_R10      11
52 #define KDIREG_R11      12
53 #define KDIREG_R12      13
54 #define KDIREG_R13      14
55 #define KDIREG_R14      15
56 #define KDIREG_R15      16
57 #define KDIREG_FSBASE   17
58 #define KDIREG_GSBASE   18
59 #define KDIREG_KGSBASE  19
60 #define KDIREG_CR2      20
61 #define KDIREG_CR3      21
62 #define KDIREG_DS       22
63 #define KDIREG_ES       23
64 #define KDIREG_FS       24
65 #define KDIREG_GS       25
66 #define KDIREG_TRAPNO   26
67 #define KDIREG_ERR      27
68 #define KDIREG_RIP      28
69 #define KDIREG_CS       29
70 #define KDIREG_RFLAGS   30
71 #define KDIREG_RSP      31
72 #define KDIREG_SS       32
62 #define KDIREG_DS       20
63 #define KDIREG_ES       21
64 #define KDIREG_FS       22
65 #define KDIREG_GS       23
66 #define KDIREG_TRAPNO   24
67 #define KDIREG_ERR      25
68 #define KDIREG_RIP      26
69 #define KDIREG_CS       27
70 #define KDIREG_RFLAGS   28
71 #define KDIREG_RSP      29
72 #define KDIREG_SS       30

74 #define KDIREG_NGREG    (KDIREG_SS + 1)

76 #define KDIREG_PC       KDIREG_RIP
77 #define KDIREG_SP       KDIREG_RSP
78 #define KDIREG_FP       KDIREG_RBP

78 #ifdef _ASM

80 /* Patch point for MSR clearing. */
81 #define KDI_MSR_PATCH \
82         nop; nop; nop; nop; \
83         nop; nop; nop; nop; \
84         nop; nop; nop; nop; \
85         nop; nop; nop; nop; \
86         nop

88 #endif  /* _ASM */

90 #define KDI_MSR_PATCHOFF        8       /* bytes of code before patch point */
91 #define KDI_MSR_PATCHSZ         17      /* bytes in KDI_MSR_PATCH, above */

80 #ifdef __cplusplus
81 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    2038 Fri Apr  6 17:25:09 2018
new/usr/src/uts/intel/asm/htable.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License, Version 1.0 only
 * (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 2004 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

#ifndef _ASM_HTABLE_H
#define _ASM_HTABLE_H

#include <sys/ccompile.h>
#include <sys/types.h>

#ifdef  __cplusplus
extern "C" {
#endif

#if !defined(__lint) && defined(__GNUC__)

#if defined(__i386) || defined(__amd64)

/*
 * This set of atomic operations are designed primarily
 * for some ia32 hat layer operations.
 */

extern __GNU_INLINE void
atomic_orb(uint8_t *addr, uint8_t value)
{
        __asm__ __volatile__(
            "lock; orb %%dl,%0"
            : "=m" (*addr)
            : "d" (value), "m" (*addr)
            : "cc");
}
_____unchanged_portion_omitted_

#endif  /* __i386 || __amd64 */

#endif  /* !__lint && __GNUC__ */

#ifdef  __cplusplus
}
#endif

#endif  /* _ASM_HTABLE_H */
```



```
#endif  /* __i386 || __amd64 */

#endif  /* !__lint && __GNUC__ */

#ifdef  __cplusplus
}
#endif

#endif  /* _ASM_HTABLE_H */
```

```
**********************************************************
    2393 Fri Apr  6 17:25:10 2018
new/usr/src/uts/intel/asm/mmu.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 _ASM_MMU_H
#define _ASM_MMU_H

#include <sys/ccompile.h>
#include <sys/types.h>

#ifdef  __cplusplus
extern "C" {
#endif

#if defined(__GNUC__)
#if defined(__GNUC__) && !defined(__xpv)

#if !defined(__xpv)
#if defined(__amd64)

extern __GNU_INLINE ulong_t
getcr3(void)
{
        uint64_t value;

        __asm__ __volatile__(
            "movq %%cr3, %0"
            : "=r" (value));
        return (value);
}
_____unchanged_portion_omitted_

extern __GNU_INLINE void
reload_cr3(void)
{
        setcr3(getcr3());
```



```
}

#elif defined(__i386)

extern __GNU_INLINE ulong_t
getcr4(void)
getcr3(void)
{
        uint64_t value;
        uint32_t value;

        __asm__ __volatile__(
            "movq %%cr4, %0"
            "movl %%cr3, %0"
            : "=r" (value));
        return (value);
}

extern __GNU_INLINE void
setcr4(ulong_t value)
setcr3(ulong_t value)
{
        __asm__ __volatile__(
            "movq %0, %%cr4"
            "movl %0, %%cr3"
            : /* no output */
            : "r" (value));
}
_____unchanged_portion_omitted_

/*
 * We clobber memory: we're not writing anything, but we don't want to
 * potentially get re-ordered beyond the TLB flush.
 */
extern __GNU_INLINE void
invpcid_insn(uint64_t type, uint64_t pcid, uintptr_t addr)
{
        uint64_t pcid_desc[2] = { pcid, addr };
        __asm__ __volatile__(
            "invpcid %0, %1"
            : /* no output */
            : "m" (*pcid_desc), "r" (type)
            : "memory");
}
#endif

#endif /* !__xpv */
#endif /* __GNUC__ && !__xpv */

extern __GNU_INLINE void
mmu_invlpg(caddr_t addr)
{
        __asm__ __volatile__(
            "invlpg %0"
            : "=m" (*addr)
            : "m" (*addr));
}

#endif /* __GNUC__ */

#ifdef __cplusplus
}
_____unchanged_portion_omitted_
```

```
**********************************************************
   34750 Fri Apr  6 17:25:10 2018
new/usr/src/uts/intel/ia32/ml/exception.s
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
```

```
   1 /*
   2  * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
   3  * Copyright (c) 2013, 2014 by Delphix. All rights reserved.
   4  * Copyright (c) 2018 Joyent, Inc.
   4  * Copyright (c) 2017 Joyent, Inc.
   5  */

   7 /*
   8  * Copyright (c) 1989, 1990 William F. Jolitz.
   9  * Copyright (c) 1990 The Regents of the University of California.
  10  * All rights reserved.
  11  *
  12  * Redistribution and use in source and binary forms, with or without
  13  * modification, are permitted provided that the following conditions
  14  * are met:
  15  * 1. Redistributions of source code must retain the above copyright
  16  *    notice, this list of conditions and the following disclaimer.
  17  * 2. Redistributions in binary form must reproduce the above copyright
  18  *    notice, this list of conditions and the following disclaimer in the
  19  *    documentation and/or other materials provided with the distribution.
  20  * 3. All advertising materials mentioning features or use of this software
  21  *    must display the following acknowledgement:
  22  *      This product includes software developed by the University of
  23  *      California, Berkeley and its contributors.
  24  * 4. Neither the name of the University nor the names of its contributors
  25  *    may be used to endorse or promote products derived from this software
  26  *    without specific prior written permission.
  27  *
  28  * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ''AS IS'' AND
  29  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  30  * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  31  * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
  32  * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
  33  * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
  34  * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
  35  * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
  36  * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
  37  * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
  38  * SUCH DAMAGE.
  39  *
  40  * $FreeBSD: src/sys/amd64/amd64/exception.S,v 1.113 2003/10/15 02:04:52 peter E
  41  */

  43 #include <sys/asm_linkage.h>
  44 #include <sys/asm_misc.h>
  45 #include <sys/trap.h>
  46 #include <sys/psw.h>
  47 #include <sys/regset.h>
  48 #include <sys/privregs.h>
  49 #include <sys/dtrace.h>
  50 #include <sys/x86_archext.h>
  51 #include <sys/traptrace.h>
  52 #include <sys/machparam.h>

  54 /*
  55  * only one routine in this file is interesting to lint
  56  */

  58 #if defined(__lint)
```



```
  60 void
  61 ndptrap_frstor(void)
  62 {}

  64 #else

  66 #include "assym.h"

  68 /*
  69  * push $0 on stack for traps that do not
  70  * generate an error code. This is so the rest
  71  * of the kernel can expect a consistent stack
  72  * from from any exception.
  73  *
  74  * Note that for all exceptions for amd64
  75  * %r11 and %rcx are on the stack. Just pop
  76  * them back into their appropriate registers and let
  77  * it get saved as is running native.
  78  */

  80 #if defined(__xpv) && defined(__amd64)

  82 #define NPTRAP_NOERR(trapno)    \
  83         pushq   $0;             \
  84         pushq   $trapno

  86 #define TRAP_NOERR(trapno)      \
  87         XPV_TRAP_POP;           \
  88         NPTRAP_NOERR(trapno)

  90 /*
  91  * error code already pushed by hw
  92  * onto stack.
  93  */
  94 #define TRAP_ERR(trapno)        \
  95         XPV_TRAP_POP;           \
  96         pushq   $trapno

  98 #else /* __xpv && __amd64 */

 100 #define TRAP_NOERR(trapno)      \
 101         push    $0;             \
 102         push    $trapno

 104 #define NPTRAP_NOERR(trapno) TRAP_NOERR(trapno)

 106 /*
 107  * error code already pushed by hw
 108  * onto stack.
 109  */
 110 #define TRAP_ERR(trapno)        \
 111         push    $trapno

 113 #endif  /* __xpv && __amd64 */

 115         /*
 116          * These are the stacks used on cpu0 for taking double faults,
 117          * NMIs and MCEs (the latter two only on amd64 where we have IST).
 118          *
 119          * We define them here instead of in a C file so that we can page-align
 120          * them (gcc won't do that in a .c file).
 121          */
 122         .data
 123         DGDEF3(dblfault_stack0, DEFAULTSTKSZ, MMU_PAGESIZE)
 124         .fill   DEFAULTSTKSZ, 1, 0
```

```
125         DGDEF3(nmi_stack0, DEFAULTSTKSZ, MMU_PAGESIZE)
126         .fill   DEFAULTSTKSZ, 1, 0
127         DGDEF3(mce_stack0, DEFAULTSTKSZ, MMU_PAGESIZE)
128         .fill   DEFAULTSTKSZ, 1, 0

130         /*
131          * #DE
132          */
133         ENTRY_NP(div0trap)
134         TRAP_NOERR(T_ZERODIV)   /* $0 */
135         jmp     cmntrap
136         SET_SIZE(div0trap)

138         /*
139          * #DB
140          *
141          * Fetch %dr6 and clear it, handing off the value to the
142          * cmntrap code in %r15/%esi
143          */
144         ENTRY_NP(dbgtrap)
145         TRAP_NOERR(T_SGLSTP)    /* $1 */

147 #if defined(__amd64)
148 #if !defined(__xpv)             /* no sysenter support yet */
149         /*
150          * If we get here as a result of single-stepping a sysenter
151          * instruction, we suddenly find ourselves taking a #db
152          * in kernel mode -before- we've swapgs'ed.  So before we can
153          * take the trap, we do the swapgs here, and fix the return
154          * %rip in trap() so that we return immediately after the
155          * swapgs in the sysenter handler to avoid doing the swapgs again.
156          *
157          * Nobody said that the design of sysenter was particularly
158          * elegant, did they?
159          */

161         pushq   %r11

163         /*
164          * At this point the stack looks like this:
165          *
166          * (high address)       r_ss
167          *                      r_rsp
168          *                      r_rfl
169          *                      r_cs
170          *                      r_rip           <-- %rsp + 24
171          *                      r_err           <-- %rsp + 16
172          *                      r_trapno        <-- %rsp + 8
173          * (low address)        %r11            <-- %rsp
174          */
175         leaq    sys_sysenter(%rip), %r11
176         cmpq    %r11, 24(%rsp)  /* Compare to saved r_rip on the stack */
177         je      1f
178         leaq    brand_sys_sysenter(%rip), %r11
179         cmpq    %r11, 24(%rsp)  /* Compare to saved r_rip on the stack */
180         je      1f
181         leaq    tr_sys_sysenter(%rip), %r11
182         cmpq    %r11, 24(%rsp)
183         je      1f
184         leaq    tr_brand_sys_sysenter(%rip), %r11
185         cmpq    %r11, 24(%rsp)
186         jne     2f
187 1:      SWAPGS
188 2:      popq    %r11
189 #endif  /* !__xpv */
```



```
191         INTR_PUSH
192 #if defined(__xpv)
193         movl    $6, %edi
194         call    kdi_dreg_get
195         movq    %rax, %r15              /* %db6 -> %r15 */
196         movl    $6, %edi
197         movl    $0, %esi
198         call    kdi_dreg_set            /* 0 -> %db6 */
199 #else
200         movq    %db6, %r15
201         xorl    %eax, %eax
202         movq    %rax, %db6
203 #endif

205 #elif defined(__i386)

207         INTR_PUSH
208 #if defined(__xpv)
209         pushl   $6
210         call    kdi_dreg_get
211         addl    $4, %esp
212         movl    %eax, %esi              /* %dr6 -> %esi */
213         pushl   $0
214         pushl   $6
215         call    kdi_dreg_set            /* 0 -> %dr6 */
216         addl    $8, %esp
217 #else
218         movl    %db6, %esi
219         xorl    %eax, %eax
220         movl    %eax, %db6
221 #endif
222 #endif  /* __i386 */

224         jmp     cmntrap_pushed
225         SET_SIZE(dbgtrap)

227 #if defined(__amd64)
228 #if !defined(__xpv)

230 /*
231  * Macro to set the gsbase or kgsbase to the address of the struct cpu
232  * for this processor.  If we came from userland, set kgsbase else
233  * set gsbase.  We find the proper cpu struct by looping through
234  * the cpu structs for all processors till we find a match for the gdt
235  * of the trapping processor.  The stack is expected to be pointing at
236  * the standard regs pushed by hardware on a trap (plus error code and trapno).
237  *
238  * It's ok for us to clobber gsbase here (and possibly end up with both gsbase
239  * and kgsbase set to the same value) because we're not going back the normal
240  * way out of here (via IRET). Where we're going, we don't need no user %gs.
241  */
242 #define SET_CPU_GSBASE                                                  \
243         subq    $REGOFF_TRAPNO, %rsp;   /* save regs */                 \
244         movq    %rax, REGOFF_RAX(%rsp);                                 \
245         movq    %rbx, REGOFF_RBX(%rsp);                                 \
246         movq    %rcx, REGOFF_RCX(%rsp);                                 \
247         movq    %rdx, REGOFF_RDX(%rsp);                                 \
248         movq    %rbp, REGOFF_RBP(%rsp);                                 \
249         movq    %rsp, %rbp;                                             \
250         subq    $16, %rsp;              /* space for gdt */             \
251         sgdt    6(%rsp);                                                \
252         movq    8(%rsp), %rcx;          /* %rcx has gdt to match */     \
253         xorl    %ebx, %ebx;             /* loop index */                \
254         leaq    cpu(%rip), %rdx;        /* cpu pointer array */         \
255 1:                                                                      \
256         movq    (%rdx, %rbx, CLONGSIZE), %rax;  /* get cpu[i] */        \
```

```
 257         cmpq    $0x0, %rax;             /* cpu[i] == NULL ? */          \
 258         je      2f;                     /* yes, continue */             \
 259         cmpq    %rcx, CPU_GDT(%rax);    /* gdt == cpu[i]->cpu_gdt ? */  \
 260         je      3f;                     /* yes, go set gsbase */        \
 261 2:                                                                      \
 262         incl    %ebx;                   /* i++ */                       \
 263         cmpl    $NCPU, %ebx;            /* i < NCPU ? */                \
 264         jb      1b;                     /* yes, loop */                 \
 265 /* XXX BIG trouble if we fall thru here.  We didn't find a gdt match */ \
 266 3:                                                                      \
 267         movl    $MSR_AMD_KGSBASE, %ecx;                                 \
 268         cmpw    $KCS_SEL, REGOFF_CS(%rbp); /* trap from kernel? */      \
 269         jne     4f;                     /* no, go set KGSBASE */        \
 270         movl    $MSR_AMD_GSBASE, %ecx;  /* yes, set GSBASE */           \
 271         mfence;                         /* OPTERON_ERRATUM_88 */        \
 272 4:                                                                      \
 273         movq    %rax, %rdx;             /* write base register */       \
 274         shrq    $32, %rdx;                                              \
 275         wrmsr;                                                          \
 276         movq    REGOFF_RDX(%rbp), %rdx; /* restore regs */              \
 277         movq    REGOFF_RCX(%rbp), %rcx;                                 \
 278         movq    REGOFF_RBX(%rbp), %rbx;                                 \
 279         movq    REGOFF_RAX(%rbp), %rax;                                 \
 280         movq    %rbp, %rsp;                                             \
 281         movq    REGOFF_RBP(%rsp), %rbp;                                 \
 282         addq    $REGOFF_TRAPNO, %rsp    /* pop stack */

 284 #else   /* __xpv */

 286 #define SET_CPU_GSBASE  /* noop on the hypervisor */

 288 #endif  /* __xpv */
 289 #endif  /* __amd64 */


 292 #if defined(__amd64)

 294         /*
 295          * #NMI
 296          *
 297          * XXPV: See 6532669.
 298          */
 299         ENTRY_NP(nmiint)
 300         TRAP_NOERR(T_NMIFLT)    /* $2 */

 302         SET_CPU_GSBASE

 304         /*
 305          * Save all registers and setup segment registers
 306          * with kernel selectors.
 307          */
 308         INTR_PUSH
 309         INTGATE_INIT_KERNEL_FLAGS

 311         TRACE_PTR(%r12, %rax, %eax, %rdx, $TT_TRAP)
 312         TRACE_REGS(%r12, %rsp, %rax, %rbx)
 313         TRACE_STAMP(%r12)

 315         movq    %rsp, %rbp

 317         movq    %rbp, %rdi
 318         call    av_dispatch_nmivect

 320         INTR_POP
 321         jmp     tr_iret_auto
 297         IRET
```



```
 322         /*NOTREACHED*/
 323         SET_SIZE(nmiint)
_____unchanged_portion_omitted_

 402 #if defined(__amd64)

 404         ENTRY_NP(invoptrap)

 406         XPV_TRAP_POP

 408         cmpw    $KCS_SEL, 8(%rsp)
 409         jne     ud_user

 411 #if defined(__xpv)
 412         movb    $0, 12(%rsp)            /* clear saved upcall_mask from %cs */
 413 #endif
 414         push    $0                      /* error code -- zero for #UD */
 415 ud_kernel:
 416         push    $0xdddd                 /* a dummy trap number */
 417         INTR_PUSH
 418         movq    REGOFF_RIP(%rsp), %rdi
 419         movq    REGOFF_RSP(%rsp), %rsi
 420         movq    REGOFF_RAX(%rsp), %rdx
 421         pushq   (%rsi)
 422         movq    %rsp, %rsi
 423         subq    $8, %rsp
 424         call    dtrace_invop
 425         ALTENTRY(dtrace_invop_callsite)
 426         addq    $16, %rsp
 427         cmpl    $DTRACE_INVOP_PUSHL_EBP, %eax
 428         je      ud_push
 429         cmpl    $DTRACE_INVOP_LEAVE, %eax
 430         je      ud_leave
 431         cmpl    $DTRACE_INVOP_NOP, %eax
 432         je      ud_nop
 433         cmpl    $DTRACE_INVOP_RET, %eax
 434         je      ud_ret
 435         jmp     ud_trap

 437 ud_push:
 438         /*
 439          * We must emulate a "pushq %rbp".  To do this, we pull the stack
 440          * down 8 bytes, and then store the base pointer.
 441          */
 442         INTR_POP
 443         subq    $16, %rsp               /* make room for %rbp */
 444         pushq   %rax                    /* push temp */
 445         movq    24(%rsp), %rax          /* load calling RIP */
 446         addq    $1, %rax                /* increment over trapping instr */
 447         movq    %rax, 8(%rsp)           /* store calling RIP */
 448         movq    32(%rsp), %rax          /* load calling CS */
 449         movq    %rax, 16(%rsp)          /* store calling CS */
 450         movq    40(%rsp), %rax          /* load calling RFLAGS */
 451         movq    %rax, 24(%rsp)          /* store calling RFLAGS */
 452         movq    48(%rsp), %rax          /* load calling RSP */
 453         subq    $8, %rax                /* make room for %rbp */
 454         movq    %rax, 32(%rsp)          /* store calling RSP */
 455         movq    56(%rsp), %rax          /* load calling SS */
 456         movq    %rax, 40(%rsp)          /* store calling SS */
 457         movq    32(%rsp), %rax          /* reload calling RSP */
 458         movq    %rbp, (%rax)            /* store %rbp there */
 459         popq    %rax                    /* pop off temp */
 460         jmp     tr_iret_kernel          /* return from interrupt */
 436         IRET                            /* return from interrupt */
 461         /*NOTREACHED*/
```

```
463 ud_leave:
464         /*
465          * We must emulate a "leave", which is the same as a "movq %rbp, %rsp"
466          * followed by a "popq %rbp".  This is quite a bit simpler on amd64
467          * than it is on i386 -- we can exploit the fact that the %rsp is
468          * explicitly saved to effect the pop without having to reshuffle
469          * the other data pushed for the trap.
470          */
471         INTR_POP
472         pushq   %rax                    /* push temp */
473         movq    8(%rsp), %rax           /* load calling RIP */
474         addq    $1, %rax                /* increment over trapping instr */
475         movq    %rax, 8(%rsp)           /* store calling RIP */
476         movq    (%rbp), %rax            /* get new %rbp */
477         addq    $8, %rbp                /* adjust new %rsp */
478         movq    %rbp, 32(%rsp)          /* store new %rsp */
479         movq    %rax, %rbp              /* set new %rbp */
480         popq    %rax                    /* pop off temp */
481         jmp     tr_iret_kernel          /* return from interrupt */
457         IRET                            /* return from interrupt */
482         /*NOTREACHED*/

484 ud_nop:
485         /*
486          * We must emulate a "nop".  This is obviously not hard:  we need only
487          * advance the %rip by one.
488          */
489         INTR_POP
490         incq    (%rsp)
491         jmp     tr_iret_kernel
467         IRET
492         /*NOTREACHED*/

494 ud_ret:
495         INTR_POP
496         pushq   %rax                    /* push temp */
497         movq    32(%rsp), %rax          /* load %rsp */
498         movq    (%rax), %rax            /* load calling RIP */
499         movq    %rax, 8(%rsp)           /* store calling RIP */
500         addq    $8, 32(%rsp)            /* adjust new %rsp */
501         popq    %rax                    /* pop off temp */
502         jmp     tr_iret_kernel          /* return from interrupt */
478         IRET                            /* return from interrupt */
503         /*NOTREACHED*/

505 ud_trap:
506         /*
507          * We're going to let the kernel handle this as a normal #UD.  If,
508          * however, we came through #BP and are spoofing #UD (in this case,
509          * the stored error value will be non-zero), we need to de-spoof
510          * the trap by incrementing %rip and pushing T_BPTFLT.
511          */
512         cmpq    $0, REGOFF_ERR(%rsp)
513         je      ud_ud
514         incq    REGOFF_RIP(%rsp)
515         addq    $REGOFF_RIP, %rsp
516         NPTRAP_NOERR(T_BPTFLT)  /* $3 */
517         jmp     cmntrap

519 ud_ud:
520         addq    $REGOFF_RIP, %rsp
521 ud_user:
522         NPTRAP_NOERR(T_ILLINST)
523         jmp     cmntrap
524         SET_SIZE(invoptrap)
______unchanged_portion_omitted_
```



```
718 #else   /* __xpv */

720         ENTRY_NP(ndptrap)
721         /*
722          * We want to do this quickly as every lwp using fp will take this
723          * after a context switch -- we do the frequent path in ndptrap_frstor
724          * below; for all other cases, we let the trap code handle it
725          */
726         pushq   %rax
727         pushq   %rbx
728         cmpw    $KCS_SEL, 24(%rsp)      /* did we come from kernel mode? */
729         jne     1f
730         LOADCPU(%rax)                   /* if yes, don't swapgs */
731         jmp     2f
732 1:
733         SWAPGS                          /* if from user, need swapgs */
734         LOADCPU(%rax)
735         SWAPGS
736 2:
737         /*
738          * Xrstor needs to use edx as part of its flag.
739          * NOTE: have to push rdx after "cmpw ...24(%rsp)", otherwise rsp+$24
740          * will not point to CS.
741          */
742         pushq   %rdx
743         cmpl    $0, fpu_exists(%rip)
744         je      .handle_in_trap         /* let trap handle no fp case */
745         movq    CPU_THREAD(%rax), %rbx  /* %rbx = curthread */
746         movl    $FPU_EN, %eax
747         movq    T_LWP(%rbx), %rbx       /* %rbx = lwp */
748         testq   %rbx, %rbx
749         jz      .handle_in_trap         /* should not happen? */
750 #if LWP_PCB_FPU != 0
751         addq    $LWP_PCB_FPU, %rbx      /* &lwp->lwp_pcb.pcb_fpu */
752 #endif
753         testl   %eax, PCB_FPU_FLAGS(%rbx)
754         jz      .handle_in_trap         /* must be the first fault */
755         clts
756         andl    $_BITNOT(FPU_VALID), PCB_FPU_FLAGS(%rbx)
757 #if FPU_CTX_FPU_REGS != 0
758         addq    $FPU_CTX_FPU_REGS, %rbx
759 #endif

761         movl    FPU_CTX_FPU_XSAVE_MASK(%rbx), %eax      /* for xrstor */
762         movl    FPU_CTX_FPU_XSAVE_MASK+4(%rbx), %edx    /* for xrstor */

764         /*
765          * the label below is used in trap.c to detect FP faults in
766          * kernel due to user fault.
767          */
768         ALTENTRY(ndptrap_frstor)
769         movq (%rbx), %rbx               /* fpu_regs.kfpu_u.kfpu_XX pointer */
770         .globl  _patch_xrstorq_rbx
771 _patch_xrstorq_rbx:
772         fxrstorq (%rbx)
773         popq    %rdx
774         popq    %rbx
775         popq    %rax
776         jmp     tr_iret_auto
752         IRET
777         /*NOTREACHED*/

779 .handle_in_trap:
780         popq    %rdx
781         popq    %rbx
```

```
 782         popq    %rax
 783         TRAP_NOERR(T_NOEXTFLT)  /* $7 */
 784         jmp     cmninttrap
 785         SET_SIZE(ndptrap_frstor)
______unchanged_portion_omitted_

1151 #endif  /* !__amd64 */

1153         ENTRY_NP(resvtrap)
1154         TRAP_NOERR(T_RESVTRAP)  /* (reserved)  */
1130         TRAP_NOERR(15)          /* (reserved)  */
1155         jmp     cmntrap
1156         SET_SIZE(resvtrap)
______unchanged_portion_omitted_

1233         ENTRY_NP(invaltrap)
1234         TRAP_NOERR(T_INVALTRAP) /* very invalid */
1210         TRAP_NOERR(30)          /* very invalid */
1235         jmp     cmntrap
1236         SET_SIZE(invaltrap)

1214         ENTRY_NP(invalint)
1215         TRAP_NOERR(31)          /* even more so */
1216         jmp     cmnint
1217         SET_SIZE(invalint)

1238         .globl  fasttable

1240 #if defined(__amd64)

1242         ENTRY_NP(fasttrap)
1243         cmpl    $T_LASTFAST, %eax
1244         ja      1f
1245         orl     %eax, %eax      /* (zero extend top 32-bits) */
1246         leaq    fasttable(%rip), %r11
1247         leaq    (%r11, %rax, CLONGSIZE), %r11
1248         jmp     *(%r11)
1249 1:
1250         /*
1251          * Fast syscall number was illegal.  Make it look
1252          * as if the INT failed.  Modify %rip to point before the
1253          * INT, push the expected error code and fake a GP fault.
1254          *
1255          * XXX Why make the error code be offset into idt + 1?
1256          * Instead we should push a real (soft?) error code
1257          * on the stack and #gp handler could know about fasttraps?
1258          */
1259         XPV_TRAP_POP

1261         subq    $2, (%rsp)      /* XXX int insn 2-bytes */
1262         pushq   $_CONST(_MUL(T_FASTTRAP, GATE_DESC_SIZE) + 2)

1264 #if defined(__xpv)
1265         pushq   %r11
1266         pushq   %rcx
1267 #endif
1268         jmp     gptrap
1269         SET_SIZE(fasttrap)
______unchanged_portion_omitted_

1299 #if defined(__amd64)

1301         /*
1302          * RFLAGS 24 bytes up the stack from %rsp.
1303          * XXX a constant would be nicer.
1304          */
```



```
1305         ENTRY_NP(fast_null)
1306         XPV_TRAP_POP
1307         orq     $PS_C, 24(%rsp) /* set carry bit in user flags */
1308         jmp     tr_iret_auto
1289         IRET
1309         /*NOTREACHED*/
1310         SET_SIZE(fast_null)
______unchanged_portion_omitted_
```

```
**********************************************************
   81879 Fri Apr  6 17:25:10 2018
new/usr/src/uts/intel/ia32/ml/i86_subr.s
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
  24  * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
  25  * Copyright (c) 2014 by Delphix. All rights reserved.
  26  * Copyright 2018 Joyent, Inc.
  26  * Copyright 2016 Joyent, Inc.
  27  */

  29 /*
  30  *  Copyright (c) 1990, 1991 UNIX System Laboratories, Inc.
  31  *  Copyright (c) 1984, 1986, 1987, 1988, 1989, 1990 AT&T
  32  *    All Rights Reserved
  33  */

  35 /*
  36  * Copyright (c) 2009, Intel Corporation.
  37  * All rights reserved.
  38  */

  40 /*
  41  * General assembly language routines.
  42  * It is the intent of this file to contain routines that are
  43  * independent of the specific kernel architecture, and those that are
  44  * common across kernel architectures.
  45  * As architectures diverge, and implementations of specific
  46  * architecture-dependent routines change, the routines should be moved
  47  * from this file into the respective ../`arch -k`/subr.s file.
  48  */

  50 #include <sys/asm_linkage.h>
  51 #include <sys/asm_misc.h>
  52 #include <sys/panic.h>
  53 #include <sys/ontrap.h>
  54 #include <sys/regset.h>
  55 #include <sys/privregs.h>
  56 #include <sys/reboot.h>
  57 #include <sys/psw.h>
  58 #include <sys/x86_archext.h>
```



```
  60 #if defined(__lint)
  61 #include <sys/types.h>
  62 #include <sys/systm.h>
  63 #include <sys/thread.h>
  64 #include <sys/archsystm.h>
  65 #include <sys/byteorder.h>
  66 #include <sys/dtrace.h>
  67 #include <sys/ftrace.h>
  68 #else   /* __lint */
  69 #include "assym.h"
  70 #endif  /* __lint */
  71 #include <sys/dditypes.h>

  73 /*
  74  * on_fault()
  75  *
  76  * Catch lofault faults. Like setjmp except it returns one
  77  * if code following causes uncorrectable fault. Turned off
  78  * by calling no_fault(). Note that while under on_fault(),
  79  * SMAP is disabled. For more information see
  80  * uts/intel/ia32/ml/copy.s.
  81  */

  83 #if defined(__lint)

  85 /* ARGSUSED */
  86 int
  87 on_fault(label_t *ljb)
  88 { return (0); }

  90 void
  91 no_fault(void)
  92 {}

  94 #else   /* __lint */

  96 #if defined(__amd64)

  98         ENTRY(on_fault)
  99         movq    %gs:CPU_THREAD, %rsi
 100         leaq    catch_fault(%rip), %rdx
 101         movq    %rdi, T_ONFAULT(%rsi)           /* jumpbuf in t_onfault */
 102         movq    %rdx, T_LOFAULT(%rsi)           /* catch_fault in t_lofault */
 103         call    smap_disable                    /* allow user accesses */
 104         jmp     setjmp                          /* let setjmp do the rest */

 106 catch_fault:
 107         movq    %gs:CPU_THREAD, %rsi
 108         movq    T_ONFAULT(%rsi), %rdi           /* address of save area */
 109         xorl    %eax, %eax
 110         movq    %rax, T_ONFAULT(%rsi)           /* turn off onfault */
 111         movq    %rax, T_LOFAULT(%rsi)           /* turn off lofault */
 112         call    smap_enable                     /* disallow user accesses */
 113         jmp     longjmp                         /* let longjmp do the rest */
 114         SET_SIZE(on_fault)
_______unchanged_portion_omitted_

 428 #endif  /* __i386 */
 429 #endif  /* __lint */

 431 /*
 432  * Invalidate a single page table entry in the TLB
 433  */

 435 #if defined(__lint)
```

```
 437 /* ARGSUSED */
 438 void
 439 mmu_invlpg(caddr_t m)
 439 mmu_tlbflush_entry(caddr_t m)
 440 {}

 442 #else   /* __lint */

 444         ENTRY(mmu_invlpg)
 444 #if defined(__amd64)

 446         ENTRY(mmu_tlbflush_entry)
 445         invlpg  (%rdi)
 446         ret
 447         SET_SIZE(mmu_invlpg)
 449         SET_SIZE(mmu_tlbflush_entry)

 451 #elif defined(__i386)

 453         ENTRY(mmu_tlbflush_entry)
 454         movl    4(%esp), %eax
 455         invlpg  (%eax)
 456         ret
 457         SET_SIZE(mmu_tlbflush_entry)

 459 #endif  /* __i386 */
 449 #endif  /* __lint */


 452 /*
 453  * Get/Set the value of various control registers
 454  */

 456 #if defined(__lint)

 458 ulong_t
 459 getcr0(void)
 460 { return (0); }

 462 /* ARGSUSED */
 463 void
 464 setcr0(ulong_t value)
 465 {}

 467 ulong_t
 468 getcr2(void)
 469 { return (0); }

 471 ulong_t
 472 getcr3(void)
 473 { return (0); }

 475 #if !defined(__xpv)
 476 /* ARGSUSED */
 477 void
 478 setcr3(ulong_t val)
 479 {}

 481 void
 482 reload_cr3(void)
 483 {}
 484 #endif

 486 ulong_t
 487 getcr4(void)
```



```
 488 { return (0); }

 490 /* ARGSUSED */
 491 void
 492 setcr4(ulong_t val)
 493 {}

 495 #if defined(__amd64)

 497 ulong_t
 498 getcr8(void)
 499 { return (0); }

 501 /* ARGSUSED */
 502 void
 503 setcr8(ulong_t val)
 504 {}

 506 #endif  /* __amd64 */

 508 #else   /* __lint */

 510 #if defined(__amd64)

 512         ENTRY(getcr0)
 513         movq    %cr0, %rax
 514         ret
 515         SET_SIZE(getcr0)
______unchanged_portion_omitted_
```

```
**********************************************************
   25007 Fri Apr  6 17:25:11 2018
new/usr/src/uts/intel/ia32/ml/swtch.s
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2018 Joyent, Inc.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 */

/*
 * Process switching routines.
 */

#if defined(__lint)
#include <sys/thread.h>
#include <sys/systm.h>
#include <sys/time.h>
#else   /* __lint */
#include "assym.h"
#endif  /* __lint */

#include <sys/asm_linkage.h>
#include <sys/asm_misc.h>
#include <sys/regset.h>
#include <sys/privregs.h>
#include <sys/stack.h>
#include <sys/segments.h>
#include <sys/psw.h>

/*
 * resume(thread_id_t t);
 *
 * a thread can only run on one processor at a time. there
 * exists a window on MPs where the current thread on one
 * processor is capable of being dispatched by another processor.
 * some overlap between outgoing and incoming threads can happen
 * when they are the same thread. in this case where the threads
 * are the same, resume() on one processor will spin on the incoming
```



```
 * thread until resume() on the other processor has finished with
 * the outgoing thread.
 *
 * The MMU context changes when the resuming thread resides in a different
 * process.  Kernel threads are known by resume to reside in process 0.
 * The MMU context, therefore, only changes when resuming a thread in
 * a process different from curproc.
 *
 * resume_from_intr() is called when the thread being resumed was not
 * passivated by resume (e.g. was interrupted).  This means that the
 * resume lock is already held and that a restore context is not needed.
 * Also, the MMU context is not changed on the resume in this case.
 *
 * resume_from_zombie() is the same as resume except the calling thread
 * is a zombie and must be put on the deathrow list after the CPU is
 * off the stack.
 */

#if !defined(__lint)

#if LWP_PCB_FPU != 0
#error LWP_PCB_FPU MUST be defined as 0 for code in swtch.s to work
#endif  /* LWP_PCB_FPU != 0 */

#endif  /* !__lint */

#if defined(__amd64)

/*
 * Save non-volatile regs other than %rsp (%rbx, %rbp, and %r12 - %r15)
 *
 * The stack frame must be created before the save of %rsp so that tracebacks
 * of swtch()ed-out processes show the process as having last called swtch().
 */
#define SAVE_REGS(thread_t, retaddr)                    \
        movq    %rbp, T_RBP(thread_t);                  \
        movq    %rbx, T_RBX(thread_t);                  \
        movq    %r12, T_R12(thread_t);                  \
        movq    %r13, T_R13(thread_t);                  \
        movq    %r14, T_R14(thread_t);                  \
        movq    %r15, T_R15(thread_t);                  \
        pushq   %rbp;                                   \
        movq    %rsp, %rbp;                             \
        movq    %rsp, T_SP(thread_t);                   \
        movq    retaddr, T_PC(thread_t);                \
        movq    %rdi, %r12;                             \
        call    __dtrace_probe___sched_off__cpu

/*
 * Restore non-volatile regs other than %rsp (%rbx, %rbp, and %r12 - %r15)
 *
 * We load up %rsp from the label_t as part of the context switch, so
 * we don't repeat that here.
 *
 * We don't do a 'leave,' because reloading %rsp/%rbp from the label_t
 * already has the effect of putting the stack back the way it was when
 * we came in.
 */
#define RESTORE_REGS(scratch_reg)                       \
        movq    %gs:CPU_THREAD, scratch_reg;            \
        movq    T_RBP(scratch_reg), %rbp;               \
        movq    T_RBX(scratch_reg), %rbx;               \
        movq    T_R12(scratch_reg), %r12;               \
        movq    T_R13(scratch_reg), %r13;               \
        movq    T_R14(scratch_reg), %r14;               \
        movq    T_R15(scratch_reg), %r15
```

```
126 /*
127  * Get pointer to a thread's hat structure
128  */
129 #define GET_THREAD_HATP(hatp, thread_t, scratch_reg)    \
130         movq    T_PROCP(thread_t), hatp;                \
131         movq    P_AS(hatp), scratch_reg;                \
132         movq    A_HAT(scratch_reg), hatp

134 #define TSC_READ()                                      \
135         call    tsc_read;                               \
136         movq    %rax, %r14;

138 /*
139  * If we are resuming an interrupt thread, store a timestamp in the thread
140  * structure.  If an interrupt occurs between tsc_read() and its subsequent
141  * store, the timestamp will be stale by the time it is stored.  We can detect
142  * this by doing a compare-and-swap on the thread's timestamp, since any
143  * interrupt occurring in this window will put a new timestamp in the thread's
144  * t_intr_start field.
145  */
146 #define STORE_INTR_START(thread_t)                      \
147         testw   $T_INTR_THREAD, T_FLAGS(thread_t);      \
148         jz      1f;                                     \
149 0:                                                      \
150         TSC_READ();                                     \
151         movq    T_INTR_START(thread_t), %rax;           \
152         cmpxchgq %r14, T_INTR_START(thread_t);          \
153         jnz     0b;                                     \
154 1:

156 #elif defined (__i386)

158 /*
159  * Save non-volatile registers (%ebp, %esi, %edi and %ebx)
160  *
161  * The stack frame must be created before the save of %esp so that tracebacks
162  * of swtch()ed-out processes show the process as having last called swtch().
163  */
164 #define SAVE_REGS(thread_t, retaddr)                    \
165         movl    %ebp, T_EBP(thread_t);                  \
166         movl    %ebx, T_EBX(thread_t);                  \
167         movl    %esi, T_ESI(thread_t);                  \
168         movl    %edi, T_EDI(thread_t);                  \
169         pushl   %ebp;                                   \
170         movl    %esp, %ebp;                             \
171         movl    %esp, T_SP(thread_t);                   \
172         movl    retaddr, T_PC(thread_t);                \
173         movl    8(%ebp), %edi;                          \
174         pushl   %edi;                                   \
175         call    __dtrace_probe___sched_off__cpu;        \
176         addl    $CLONGSIZE, %esp

178 /*
179  * Restore non-volatile registers (%ebp, %esi, %edi and %ebx)
180  *
181  * We don't do a 'leave,' because reloading %rsp/%rbp from the label_t
182  * already has the effect of putting the stack back the way it was when
183  * we came in.
184  */
185 #define RESTORE_REGS(scratch_reg)                       \
186         movl    %gs:CPU_THREAD, scratch_reg;            \
187         movl    T_EBP(scratch_reg), %ebp;               \
188         movl    T_EBX(scratch_reg), %ebx;               \
189         movl    T_ESI(scratch_reg), %esi;               \
190         movl    T_EDI(scratch_reg), %edi
```



```
192 /*
193  * Get pointer to a thread's hat structure
194  */
195 #define GET_THREAD_HATP(hatp, thread_t, scratch_reg)    \
196         movl    T_PROCP(thread_t), hatp;                \
197         movl    P_AS(hatp), scratch_reg;                \
198         movl    A_HAT(scratch_reg), hatp

200 /*
201  * If we are resuming an interrupt thread, store a timestamp in the thread
202  * structure.  If an interrupt occurs between tsc_read() and its subsequent
203  * store, the timestamp will be stale by the time it is stored.  We can detect
204  * this by doing a compare-and-swap on the thread's timestamp, since any
205  * interrupt occurring in this window will put a new timestamp in the thread's
206  * t_intr_start field.
207  */
208 #define STORE_INTR_START(thread_t)                      \
209         testw   $T_INTR_THREAD, T_FLAGS(thread_t);      \
210         jz      1f;                                     \
211         pushl   %ecx;                                   \
212 0:                                                      \
213         pushl   T_INTR_START(thread_t);                 \
214         pushl   T_INTR_START+4(thread_t);               \
215         call    tsc_read;                               \
216         movl    %eax, %ebx;                             \
217         movl    %edx, %ecx;                             \
218         popl    %edx;                                   \
219         popl    %eax;                                   \
220         cmpxchg8b T_INTR_START(thread_t);               \
221         jnz     0b;                                     \
222         popl    %ecx;                                   \
223 1:

225 #endif  /* __amd64 */

227 #if defined(__lint)

229 /* ARGSUSED */
230 void
231 resume(kthread_t *t)
232 {}

234 #else   /* __lint */

236 #if defined(__amd64)

238         .global kpti_enable

240         ENTRY(resume)
241         movq    %gs:CPU_THREAD, %rax
242         leaq    resume_return(%rip), %r11

244         /*
245          * Deal with SMAP here. A thread may be switched out at any point while
246          * it is executing. The thread could be under on_fault() or it could be
247          * pre-empted while performing a copy interruption. If this happens and
248          * we're not in the context of an interrupt which happens to handle
249          * saving and restoring rflags correctly, we may lose our SMAP related
250          * state.
251          *
252          * To handle this, as part of being switched out, we first save whether
253          * or not userland access is allowed ($PS_ACHK in rflags) and store that
254          * in t_useracc on the kthread_t and unconditionally enable SMAP to
255          * protect the system.
256          *
```

```
257          * Later, when the thread finishes resuming, we potentially disable smap
258          * if PS_ACHK was present in rflags. See uts/intel/ia32/ml/copy.s for
259          * more information on rflags and SMAP.
260          */
261         pushfq
262         popq    %rsi
263         andq    $PS_ACHK, %rsi
264         movq    %rsi, T_USERACC(%rax)
265         call    smap_enable

267         /*
268          * Save non-volatile registers, and set return address for current
269          * thread to resume_return.
270          *
271          * %r12 = t (new thread) when done
272          */
273         SAVE_REGS(%rax, %r11)


276         LOADCPU(%r15)                           /* %r15 = CPU */
277         movq    CPU_THREAD(%r15), %r13          /* %r13 = curthread */

279         /*
280          * Call savectx if thread has installed context ops.
281          *
282          * Note that if we have floating point context, the save op
283          * (either fpsave_begin or fpxsave_begin) will issue the
284          * async save instruction (fnsave or fxsave respectively)
285          * that we fwait for below.
286          */
287         cmpq    $0, T_CTX(%r13)         /* should current thread savectx? */
288         je      .nosavectx              /* skip call when zero */

290         movq    %r13, %rdi              /* arg = thread pointer */
291         call    savectx                 /* call ctx ops */
292 .nosavectx:

294         /*
295          * Call savepctx if process has installed context ops.
296          */
297         movq    T_PROCP(%r13), %r14     /* %r14 = proc */
298         cmpq    $0, P_PCTX(%r14)         /* should current thread savectx? */
299         je      .nosavepctx              /* skip call when zero */

301         movq    %r14, %rdi              /* arg = proc pointer */
302         call    savepctx                 /* call ctx ops */
303 .nosavepctx:

305         /*
306          * Temporarily switch to the idle thread's stack
307          */
308         movq    CPU_IDLE_THREAD(%r15), %rax     /* idle thread pointer */

310         /*
311          * Set the idle thread as the current thread
312          */
313         movq    T_SP(%rax), %rsp        /* It is safe to set rsp */
314         movq    %rax, CPU_THREAD(%r15)

316         /*
317          * Switch in the hat context for the new thread
318          *
319          */
320         GET_THREAD_HATP(%rdi, %r12, %r11)
321         call    hat_switch
```



```
323         /*
324          * Clear and unlock previous thread's t_lock
325          * to allow it to be dispatched by another processor.
326          */
327         movb    $0, T_LOCK(%r13)

329         /*
330          * IMPORTANT: Registers at this point must be:
331          *       %r12 = new thread
332          *
333          * Here we are in the idle thread, have dropped the old thread.
334          */
335         ALTENTRY(_resume_from_idle)
336         /*
337          * spin until dispatched thread's mutex has
338          * been unlocked. this mutex is unlocked when
339          * it becomes safe for the thread to run.
340          */
341 .lock_thread_mutex:
342         lock
343         btsl    $0, T_LOCK(%r12)        /* attempt to lock new thread's mutex */
344         jnc     .thread_mutex_locked    /* got it */

346 .spin_thread_mutex:
347         pause
348         cmpb    $0, T_LOCK(%r12)        /* check mutex status */
349         jz      .lock_thread_mutex      /* clear, retry lock */
350         jmp     .spin_thread_mutex      /* still locked, spin... */

352 .thread_mutex_locked:
353         /*
354          * Fix CPU structure to indicate new running thread.
355          * Set pointer in new thread to the CPU structure.
356          */
357         LOADCPU(%r13)                   /* load current CPU pointer */
358         cmpq    %r13, T_CPU(%r12)
359         je      .setup_cpu

361         /* cp->cpu_stats.sys.cpumigrate++ */
362         incq    CPU_STATS_SYS_CPUMIGRATE(%r13)
363         movq    %r13, T_CPU(%r12)       /* set new thread's CPU pointer */

365 .setup_cpu:
366         /*
367          * Setup rsp0 (kernel stack) in TSS to curthread's saved regs
368          * structure.  If this thread doesn't have a regs structure above
369          * the stack -- that is, if lwp_stk_init() was never called for the
370          * thread -- this will set rsp0 to the wrong value, but it's harmless
371          * as it's a kernel thread, and it won't actually attempt to implicitly
372          * use the rsp0 via a privilege change.
365          * Setup rsp0 (kernel stack) in TSS to curthread's stack.
366          * (Note: Since we don't have saved 'regs' structure for all
367          *        the threads we can't easily determine if we need to
368          *        change rsp0. So, we simply change the rsp0 to bottom
369          *        of the thread stack and it will work for all cases.)
373          *
374          * Note that when we have KPTI enabled on amd64, we never use this
375          * value at all (since all the interrupts have an IST set).
371          * XX64 - Is this correct?
376          */
377         movq    CPU_TSS(%r13), %r14
378 #if !defined(__xpv)
379         cmpq    $1, kpti_enable
380         jne     1f
381         leaq    CPU_KPTI_TR_RSP(%r13), %rax
382         jmp     2f
```

```
383 1:
384         movq    T_STACK(%r12), %rax
385         addq    $REGSIZE+MINFRAME, %rax /* to the bottom of thread stack */
386 2:
387 #if !defined(__xpv)
388         movq    %rax, TSS_RSP0(%r14)
389 #else
390         movq    T_STACK(%r12), %rax
391         addq    $REGSIZE+MINFRAME, %rax /* to the bottom of thread stack */
392         movl    $KDS_SEL, %edi
393         movq    %rax, %rsi
394         call    HYPERVISOR_stack_switch
395 #endif  /* __xpv */
396
397         movq    %r12, CPU_THREAD(%r13)  /* set CPU's thread pointer */
398         mfence                          /* synchronize with mutex_exit() */
399         xorl    %ebp, %ebp              /* make $<threadlist behave better */
400         movq    T_LWP(%r12), %rax       /* set associated lwp to  */
401         movq    %rax, CPU_LWP(%r13)     /* CPU's lwp ptr */
402
403         movq    T_SP(%r12), %rsp        /* switch to outgoing thread's stack */
404         movq    T_PC(%r12), %r13        /* saved return addr */
405
406         /*
407          * Call restorectx if context ops have been installed.
408          */
409         cmpq    $0, T_CTX(%r12)         /* should resumed thread restorectx? */
410         jz      .norestorectx           /* skip call when zero */
411         movq    %r12, %rdi              /* arg = thread pointer */
412         call    restorectx              /* call ctx ops */
413 .norestorectx:
414
415         /*
416          * Call restorepctx if context ops have been installed for the proc.
417          */
418         movq    T_PROCP(%r12), %rcx
419         cmpq    $0, P_PCTX(%rcx)
420         jz      .norestorepctx
421         movq    %rcx, %rdi
422         call    restorepctx
423 .norestorepctx:
424
425         STORE_INTR_START(%r12)
426
427         /*
428          * If we came into swtch with the ability to access userland pages, go
429          * ahead and restore that fact by disabling SMAP.  Clear the indicator
430          * flag out of paranoia.
431          */
432         movq    T_USERACC(%r12), %rax   /* should we disable smap? */
433         cmpq    $0, %rax                /* skip call when zero */
434         jz      .nosmap
435         xorq    %rax, %rax
436         movq    %rax, T_USERACC(%r12)
437         call    smap_disable
438 .nosmap:
439
440         /*
441          * Restore non-volatile registers, then have spl0 return to the
442          * resuming thread's PC after first setting the priority as low as
443          * possible and blocking all interrupt threads that may be active.
444          */
445         movq    %r13, %rax      /* save return address */
446         RESTORE_REGS(%r11)
447         pushq   %rax            /* push return address for spl0() */
448         call    __dtrace_probe___sched_on__cpu
```



```
449         jmp     spl0
450
451 resume_return:
452         /*
453          * Remove stack frame created in SAVE_REGS()
454          */
455         addq    $CLONGSIZE, %rsp
456         ret
457         SET_SIZE(_resume_from_idle)
_______unchanged_portion_omitted_
```

```
**********************************************************
   38018 Fri Apr  6 17:25:11 2018
new/usr/src/uts/intel/ia32/os/desctbls.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 */

/*
 * Copyright 2018 Joyent, Inc. All rights reserved.
 * Copyright 2011 Joyent, Inc. All rights reserved.
 */

/*
 * Copyright (c) 1992 Terrence R. Lambert.
 * Copyright (c) 1990 The Regents of the University of California.
 * All rights reserved.
 *
 * This code is derived from software contributed to Berkeley by
 * William Jolitz.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
```



```
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *      from: @(#)machdep.c     7.4 (Berkeley) 6/3/91
 */

#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/tss.h>
#include <sys/segments.h>
#include <sys/trap.h>
#include <sys/cpuvar.h>
#include <sys/bootconf.h>
#include <sys/x86_archext.h>
#include <sys/controlregs.h>
#include <sys/archsystm.h>
#include <sys/machsystm.h>
#include <sys/kobj.h>
#include <sys/cmn_err.h>
#include <sys/reboot.h>
#include <sys/kdi.h>
#include <sys/mach_mmu.h>
#include <sys/systm.h>
#include <sys/note.h>

#ifdef __xpv
#include <sys/hypervisor.h>
#include <vm/as.h>
#endif

#include <sys/promif.h>
#include <sys/bootinfo.h>
#include <vm/kboot_mmu.h>
#include <vm/hat_pte.h>

/*
 * cpu0 and default tables and structures.
 */
user_desc_t     *gdt0;
#if !defined(__xpv)
desctbr_t       gdt0_default_r;
#endif

gate_desc_t     *idt0;          /* interrupt descriptor table */
#if defined(__i386)
desctbr_t       idt0_default_r;         /* describes idt0 in IDTR format */
#endif

tss_t           *ktss0;                 /* kernel task state structure */

#if defined(__i386)
tss_t           *dftss0;                /* #DF double-fault exception */
#endif  /* __i386 */

user_desc_t     zero_udesc;             /* base zero user desc native procs */
user_desc_t     null_udesc;             /* null user descriptor */
system_desc_t   null_sdesc;             /* null system descriptor */

#if defined(__amd64)
user_desc_t     zero_u32desc;           /* 32-bit compatibility procs */
#endif  /* __amd64 */
```

```
125 #if defined(__amd64)
126 user_desc_t     ucs_on;
127 user_desc_t     ucs_off;
128 user_desc_t     ucs32_on;
129 user_desc_t     ucs32_off;
130 #endif  /* __amd64 */

132 /*
133  * If the size of this is changed, you must update hat_pcp_setup() and the
134  * definitions in exception.s
135  */
136 extern char dblfault_stack0[DEFAULTSTKSZ];
137 extern char nmi_stack0[DEFAULTSTKSZ];
138 extern char mce_stack0[DEFAULTSTKSZ];
131 #pragma align   16(dblfault_stack0)
132 char            dblfault_stack0[DEFAULTSTKSZ];

140 extern void     fast_null(void);
141 extern hrtime_t get_hrtime(void);
142 extern hrtime_t gethrvtime(void);
143 extern hrtime_t get_hrestime(void);
144 extern uint64_t getlgrp(void);

146 void (*(fasttable[]))(void) = {
147         fast_null,                      /* T_FNULL routine */
148         fast_null,                      /* T_FGETFP routine (initially null) */
149         fast_null,                      /* T_FSETFP routine (initially null) */
150         (void (*)())get_hrtime,         /* T_GETHRTIME */
151         (void (*)())gethrvtime,         /* T_GETHRVTIME */
152         (void (*)())get_hrestime,       /* T_GETHRESTIME */
153         (void (*)())getlgrp             /* T_GETLGRP */
154 };
_____unchanged_portion_omitted_

315 #endif  /* __i386 */

317 /*
318  * Install gate segment descriptor for interrupt, trap, call and task gates.
319  *
320  * For 64 bit native if we have KPTI enabled, we use the IST stack mechanism on
321  * all interrupts.  We have different ISTs for each class of exceptions that are
322  * most likely to occur while handling an existing exception; while many of
323  * these are just going to panic, it’s nice not to trample on the existing
324  * exception state for debugging purposes.
325  *
326  * Normal interrupts are all redirected unconditionally to the KPTI trampoline
327  * stack space. This unifies the trampoline handling between user and kernel
328  * space (and avoids the need to touch %gs).
329  *
330  * The KDI IDT *all* uses the DBG IST: consider single stepping tr_pftrap, when
331  * we do a read from KMDB that cause another #PF.  Without its own IST, this
332  * would stomp on the kernel’s mcpu_kpti_flt frame.
333  */
334 uint_t
335 idt_vector_to_ist(uint_t vector)

315 #if defined(__amd64)

317 /*ARGSUSED*/
318 void
319 set_gatesegd(gate_desc_t *dp, void (*func)(void), selector_t sel,
320     uint_t type, uint_t dpl, uint_t vector)
336 {
337 #if defined(__xpv)
338         _NOTE(ARGUNUSED(vector));
339         return (IST_NONE);
```



```
340 #else
341         switch (vector) {
342         /* These should always use IST even without KPTI enabled. */
343         case T_DBLFLT:
344                 return (IST_DF);
345         case T_NMIFLT:
346                 return (IST_NMI);
347         case T_MCE:
348                 return (IST_MCE);
322         dp->sgd_looffset = (uintptr_t)func;
323         dp->sgd_hioffset = (uintptr_t)func >> 16;
324         dp->sgd_hi64offset = (uintptr_t)func >> (16 + 16);

350         case T_BPTFLT:
351         case T_SGLSTP:
352                 if (kpti_enable == 1) {
353                         return (IST_DBG);
354                 }
355                 return (IST_NONE);
356         case T_STKFLT:
357         case T_GPFLT:
358         case T_PGFLT:
359                 if (kpti_enable == 1) {
360                         return (IST_NESTABLE);
361                 }
362                 return (IST_NONE);
363         default:
364                 if (kpti_enable == 1) {
365                         return (IST_DEFAULT);
366                 }
367                 return (IST_NONE);
368         }
326         dp->sgd_selector =  (uint16_t)sel;

328         /*
329          * For 64 bit native we use the IST stack mechanism
330          * for double faults. All other traps use the CPL = 0
331          * (tss_rsp0) stack.
332          */
333 #if !defined(__xpv)
334         if (vector == T_DBLFLT)
335                 dp->sgd_ist = 1;
336         else
369 #endif
338                 dp->sgd_ist = 0;

340         dp->sgd_type = type;
341         dp->sgd_dpl = dpl;
342         dp->sgd_p = 1;
370 }

345 #elif defined(__i386)

347 /*ARGSUSED*/
372 void
373 set_gatesegd(gate_desc_t *dp, void (*func)(void), selector_t sel,
374     uint_t type, uint_t dpl, uint_t ist)
350     uint_t type, uint_t dpl, uint_t unused)
375 {
376         dp->sgd_looffset = (uintptr_t)func;
377         dp->sgd_hioffset = (uintptr_t)func >> 16;
378         dp->sgd_hi64offset = (uintptr_t)func >> (16 + 16);

379         dp->sgd_selector =  (uint16_t)sel;
380         dp->sgd_ist = ist;
356         dp->sgd_stkcpy = 0;     /* always zero bytes */
```

```
381         dp->sgd_type = type;
382         dp->sgd_dpl = dpl;
383         dp->sgd_p = 1;
384 }

362 #endif  /* __i386 */

386 /*
387  * Updates a single user descriptor in the the GDT of the current cpu.
388  * Caller is responsible for preventing cpu migration.
389  */

391 void
392 gdt_update_usegd(uint_t sidx, user_desc_t *udp)
393 {
394 #if defined(__xpv)

396         uint64_t dpa = CPU->cpu_m.mcpu_gdtpa + sizeof (*udp) * sidx;

398         if (HYPERVISOR_update_descriptor(pa_to_ma(dpa), *(uint64_t *)udp))
399                 panic("gdt_update_usegd: HYPERVISOR_update_descriptor");

401 #else   /* __xpv */

403         CPU->cpu_gdt[sidx] = *udp;

405 #endif  /* __xpv */
406 }
_____unchanged_portion_omitted_

922 #endif  /* __xpv */
923 #endif  /* __i386 */

925 /*
926  * Build kernel IDT.
927  *
928  * Note that for amd64 we pretty much require every gate to be an interrupt
929  * gate which blocks interrupts atomically on entry; that's because of our
930  * dependency on using 'swapgs' every time we come into the kernel to find
931  * the cpu structure. If we get interrupted just before doing that, %cs could
932  * be in kernel mode (so that the trap prolog doesn't do a swapgs), but
933  * %gsbase is really still pointing at something in userland. Bad things will
934  * ensue. We also use interrupt gates for i386 as well even though this is not
935  * required for some traps.
936  *
937  * Perhaps they should have invented a trap gate that does an atomic swapgs?
938  */
939 static void
940 init_idt_common(gate_desc_t *idt)
941 {
942         set_gatesegd(&idt[T_ZERODIV],
943             (kpti_enable == 1) ? &tr_div0trap : &div0trap,
944             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_ZERODIV));
945         set_gatesegd(&idt[T_SGLSTP],
946             (kpti_enable == 1) ? &tr_dbgtrap : &dbgtrap,
947             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_SGLSTP));
948         set_gatesegd(&idt[T_NMIFLT],
949             (kpti_enable == 1) ? &tr_nmiint : &nmiint,
950             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_NMIFLT));
951         set_gatesegd(&idt[T_BPTFLT],
952             (kpti_enable == 1) ? &tr_brktrap : &brktrap,
953             KCS_SEL, SDT_SYSIGT, TRP_UPL, idt_vector_to_ist(T_BPTFLT));
954         set_gatesegd(&idt[T_OVFLW],
955             (kpti_enable == 1) ? &tr_ovflotrap : &ovflotrap,
956             KCS_SEL, SDT_SYSIGT, TRP_UPL, idt_vector_to_ist(T_OVFLW));
957         set_gatesegd(&idt[T_BOUNDFLT],
```



```
958             (kpti_enable == 1) ? &tr_boundstrap : &boundstrap,
959             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_BOUNDFLT));
960         set_gatesegd(&idt[T_ILLINST],
961             (kpti_enable == 1) ? &tr_invoptrap : &invoptrap,
962             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_ILLINST));
963         set_gatesegd(&idt[T_NOEXTFLT],
964             (kpti_enable == 1) ? &tr_ndptrap : &ndptrap,
965             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_NOEXTFLT));
920         set_gatesegd(&idt[T_ZERODIV], &div0trap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
921             0);
922         set_gatesegd(&idt[T_SGLSTP], &dbgtrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
923             0);
924         set_gatesegd(&idt[T_NMIFLT], &nmiint, KCS_SEL, SDT_SYSIGT, TRP_KPL,
925             0);
926         set_gatesegd(&idt[T_BPTFLT], &brktrap, KCS_SEL, SDT_SYSIGT, TRP_UPL,
927             0);
928         set_gatesegd(&idt[T_OVFLW], &ovflotrap, KCS_SEL, SDT_SYSIGT, TRP_UPL,
929             0);
930         set_gatesegd(&idt[T_BOUNDFLT], &boundstrap, KCS_SEL, SDT_SYSIGT,
931             TRP_KPL, 0);
932         set_gatesegd(&idt[T_ILLINST], &invoptrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
933             0);
934         set_gatesegd(&idt[T_NOEXTFLT], &ndptrap,  KCS_SEL, SDT_SYSIGT, TRP_KPL,
935             0);

967         /*
968          * double fault handler.
969          *
970          * Note that on the hypervisor a guest does not receive #df faults.
971          * Instead a failsafe event is injected into the guest if its selectors
972          * and/or stack is in a broken state. See xen_failsafe_callback.
973          */
974 #if !defined(__xpv)
975         set_gatesegd(&idt[T_DBLFLT],
976             (kpti_enable == 1) ? &tr_syserrtrap : &syserrtrap,
977             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_DBLFLT));
945 #if defined(__amd64)

947         set_gatesegd(&idt[T_DBLFLT], &syserrtrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
948             T_DBLFLT);

950 #elif defined(__i386)

952         /*
953          * task gate required.
954          */
955         set_gatesegd(&idt[T_DBLFLT], NULL, DFTSS_SEL, SDT_SYSTASKGT, TRP_KPL,
956             0);

958 #endif  /* __i386 */
978 #endif  /* !__xpv */

980         /*
981          * T_EXTOVRFLT coprocessor-segment-overrun not supported.
982          */
983         set_gatesegd(&idt[T_TSSFLT],
984             (kpti_enable == 1) ? &tr_invtsstrap : &invtsstrap,
985             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_TSSFLT));
986         set_gatesegd(&idt[T_SEGFLT],
987             (kpti_enable == 1) ? &tr_segnptrap : &segnptrap,
988             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_SEGFLT));
989         set_gatesegd(&idt[T_STKFLT],
990             (kpti_enable == 1) ? &tr_stktrap : &stktrap,
991             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_STKFLT));
992         set_gatesegd(&idt[T_GPFLT],
993             (kpti_enable == 1) ? &tr_gptrap : &gptrap,
```

```
 994             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_GPFLT));
 995         set_gatesegd(&idt[T_PGFLT],
 996             (kpti_enable == 1) ? &tr_pftrap : &pftrap,
 997             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_PGFLT));
 998         set_gatesegd(&idt[T_EXTERRFLT],
 999             (kpti_enable == 1) ? &tr_ndperr : &ndperr,
1000             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_EXTERRFLT));
1001         set_gatesegd(&idt[T_ALIGNMENT],
1002             (kpti_enable == 1) ? &tr_achktrap : &achktrap,
1003             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_ALIGNMENT));
1004         set_gatesegd(&idt[T_MCE],
1005             (kpti_enable == 1) ? &tr_mcetrap : &mcetrap,
1006             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_MCE));
1007         set_gatesegd(&idt[T_SIMDFPE],
1008             (kpti_enable == 1) ? &tr_xmtrap : &xmtrap,
1009             KCS_SEL, SDT_SYSIGT, TRP_KPL, idt_vector_to_ist(T_SIMDFPE));

 965         set_gatesegd(&idt[T_TSSFLT], &invtsstrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 966             0);
 967         set_gatesegd(&idt[T_SEGFLT], &segnptrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 968             0);
 969         set_gatesegd(&idt[T_STKFLT], &stktrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);
 970         set_gatesegd(&idt[T_GPFLT], &gptrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);
 971         set_gatesegd(&idt[T_PGFLT], &pftrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);
 972         set_gatesegd(&idt[T_EXTERRFLT], &ndperr, KCS_SEL, SDT_SYSIGT, TRP_KPL,
 973             0);
 974         set_gatesegd(&idt[T_ALIGNMENT], &achktrap, KCS_SEL, SDT_SYSIGT,
 975             TRP_KPL, 0);
 976         set_gatesegd(&idt[T_MCE], &mcetrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);
 977         set_gatesegd(&idt[T_SIMDFPE], &xmtrap, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);

1011         /*
1012          * install fast trap handler at 210.
1013          */
1014         set_gatesegd(&idt[T_FASTTRAP],
1015             (kpti_enable == 1) ? &tr_fasttrap : &fasttrap,
1016             KCS_SEL, SDT_SYSIGT, TRP_UPL, idt_vector_to_ist(T_FASTTRAP));
 982         set_gatesegd(&idt[T_FASTTRAP], &fasttrap, KCS_SEL, SDT_SYSIGT, TRP_UPL,
 983             0);

1018         /*
1019          * System call handler.
1020          */
1021         set_gatesegd(&idt[T_SYSCALLINT],
1022             (kpti_enable == 1) ? &tr_sys_syscall_int : &sys_syscall_int,
1023             KCS_SEL, SDT_SYSIGT, TRP_UPL, idt_vector_to_ist(T_SYSCALLINT));
 988 #if defined(__amd64)
 989         set_gatesegd(&idt[T_SYSCALLINT], &sys_syscall_int, KCS_SEL, SDT_SYSIGT,
 990             TRP_UPL, 0);

 992 #elif defined(__i386)
 993         set_gatesegd(&idt[T_SYSCALLINT], &sys_call, KCS_SEL, SDT_SYSIGT,
 994             TRP_UPL, 0);
 995 #endif  /* __i386 */

1025         /*
1026          * Install the DTrace interrupt handler for the pid provider.
1027          */
1028         set_gatesegd(&idt[T_DTRACE_RET],
1029             (kpti_enable == 1) ? &tr_dtrace_ret : &dtrace_ret,
1030             KCS_SEL, SDT_SYSIGT, TRP_UPL, idt_vector_to_ist(T_DTRACE_RET));
1000         set_gatesegd(&idt[T_DTRACE_RET], &dtrace_ret, KCS_SEL,
1001             SDT_SYSIGT, TRP_UPL, 0);

1032         /*
1033          * Prepare interposing descriptor for the syscall handler
```



```
1034          * and cache copy of the default descriptor.
1035          */
1036         brand_tbl[0].ih_inum = T_SYSCALLINT;
1037         brand_tbl[0].ih_default_desc = idt0[T_SYSCALLINT];

1039         set_gatesegd(&(brand_tbl[0].ih_interp_desc),
1040             (kpti_enable == 1) ? &tr_brand_sys_syscall_int :
1041             &brand_sys_syscall_int, KCS_SEL, SDT_SYSIGT, TRP_UPL,
1042             idt_vector_to_ist(T_SYSCALLINT));
1010 #if defined(__amd64)
1011         set_gatesegd(&(brand_tbl[0].ih_interp_desc), &brand_sys_syscall_int,
1012             KCS_SEL, SDT_SYSIGT, TRP_UPL, 0);
1013 #elif defined(__i386)
1014         set_gatesegd(&(brand_tbl[0].ih_interp_desc), &brand_sys_call,
1015             KCS_SEL, SDT_SYSIGT, TRP_UPL, 0);
1016 #endif  /* __i386 */

1044         brand_tbl[1].ih_inum = 0;
1045 }
______unchanged_portion_omitted_

1055 #else   /* __xpv */

1057 static void
1058 init_idt(gate_desc_t *idt)
1059 {
1060         char    ivctname[80];
1061         void    (*ivctptr)(void);
1062         int     i;

1064         /*
1065          * Initialize entire table with 'reserved' trap and then overwrite
1066          * specific entries. T_EXTOVRFLT (9) is unsupported and reserved
1067          * since it can only be generated on a 386 processor. 15 is also
1068          * unsupported and reserved.
1069          */
1070 #if !defined(__xpv)
1071         for (i = 0; i < NIDT; i++) {
1072                 set_gatesegd(&idt[i],
1073                     (kpti_enable == 1) ? &tr_resvtrap : &resvtrap,
1074                     KCS_SEL, SDT_SYSIGT, TRP_KPL,
1075                     idt_vector_to_ist(T_RESVTRAP));
1076         }
1077 #else
1078         for (i = 0; i < NIDT; i++) {
1044         for (i = 0; i < NIDT; i++)
1079                 set_gatesegd(&idt[i], &resvtrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
1080                     IST_NONE);
1081         }
1082 #endif
1046                     0);

1084         /*
1085          * 20-31 reserved
1086          */
1087 #if !defined(__xpv)
1088         for (i = 20; i < 32; i++) {
1089                 set_gatesegd(&idt[i],
1090                     (kpti_enable == 1) ? &tr_invaltrap : &invaltrap,
1091                     KCS_SEL, SDT_SYSIGT, TRP_KPL,
1092                     idt_vector_to_ist(T_INVALTRAP));
1093         }
1094 #else
1095         for (i = 20; i < 32; i++) {
1051         for (i = 20; i < 32; i++)
1096                 set_gatesegd(&idt[i], &invaltrap, KCS_SEL, SDT_SYSIGT, TRP_KPL,
```

```
                    IST_NONE);
        }
#endif
                    0);

        /*
         * interrupts 32 - 255
         */
        for (i = 32; i < 256; i++) {
#if !defined(__xpv)
                (void) snprintf(ivctname, sizeof (ivctname),
                    (kpti_enable == 1) ? "tr_ivct%d" : "ivct%d", i);
#else
                (void) snprintf(ivctname, sizeof (ivctname), "ivct%d", i);
#endif
                ivctptr = (void (*)(void))kobj_getsymvalue(ivctname, 0);
                if (ivctptr == NULL)
                        panic("kobj_getsymvalue(%s) failed", ivctname);

                set_gatesegd(&idt[i], ivctptr, KCS_SEL, SDT_SYSIGT, TRP_KPL,
                    idt_vector_to_ist(i));
                set_gatesegd(&idt[i], ivctptr, KCS_SEL, SDT_SYSIGT, TRP_KPL, 0);
        }

        /*
         * Now install the common ones. Note that it will overlay some
         * entries installed above like T_SYSCALLINT, T_FASTTRAP etc.
         */
        init_idt_common(idt);
}
______unchanged_portion_omitted_

#if !defined(__xpv)
#if defined(__amd64)

static void
init_tss(void)
{
        extern struct cpu cpus[];
        /*
         * tss_rsp0 is dynamically filled in by resume() on each context switch.
         * All exceptions but #DF will run on the thread stack.
         * Set up the double fault stack here.
         */
        ktss0->tss_ist1 =
            (uint64_t)&dblfault_stack0[sizeof (dblfault_stack0)];

        /*
         * tss_rsp0 is dynamically filled in by resume() (in swtch.s) on each
         * context switch but it'll be overwritten with this same value anyway.
         * Set I/O bit map offset equal to size of TSS segment limit
         * for no I/O permission map. This will force all user I/O
         * instructions to generate #gp fault.
         */
        if (kpti_enable == 1) {
                ktss0->tss_rsp0 = (uint64_t)&cpus->cpu_m.mcpu_kpti.kf_tr_rsp;
        }
        ktss0->tss_bitmapbase = sizeof (*ktss0);

        /* Set up the IST stacks for double fault, NMI, MCE. */
        ktss0->tss_ist1 = (uintptr_t)&dblfault_stack0[sizeof (dblfault_stack0)];
        ktss0->tss_ist2 = (uintptr_t)&nmi_stack0[sizeof (nmi_stack0)];
        ktss0->tss_ist3 = (uintptr_t)&mce_stack0[sizeof (mce_stack0)];
        /*
         * Point %tr to descriptor for ktss0 in gdt.
         */
```



```
        wr_tsr(KTSS_SEL);
}

#elif defined(__i386)

static void
init_tss(void)
{
        /*
         * This IST stack is used for #DB,#BP (debug) interrupts (when KPTI is
         * enabled), and also for KDI (always).
         * ktss0->tss_esp dynamically filled in by resume() on each
         * context switch.
         */
        ktss0->tss_ist4 = (uint64_t)&cpus->cpu_m.mcpu_kpti_dbg.kf_tr_rsp;
        ktss0->tss_ss0  = KDS_SEL;
        ktss0->tss_eip  = (uint32_t)_start;
        ktss0->tss_ds   = ktss0->tss_es = ktss0->tss_ss = KDS_SEL;
        ktss0->tss_cs   = KCS_SEL;
        ktss0->tss_fs   = KFS_SEL;
        ktss0->tss_gs   = KGS_SEL;
        ktss0->tss_ldt  = ULDT_SEL;

        if (kpti_enable == 1) {
                /* This IST stack is used for #GP,#PF,#SS (fault) interrupts. */
                ktss0->tss_ist5 =
                    (uint64_t)&cpus->cpu_m.mcpu_kpti_flt.kf_tr_rsp;
        /*
         * Initialize double fault tss.
         */
        dftss0->tss_esp0 = (uint32_t)&dblfault_stack0[sizeof (dblfault_stack0)];
        dftss0->tss_ss0 = KDS_SEL;

                /* This IST stack is used for all other intrs (for KPTI). */
                ktss0->tss_ist6 = (uint64_t)&cpus->cpu_m.mcpu_kpti.kf_tr_rsp;
        }
        /*
         * tss_cr3 will get initialized in hat_kern_setup() once our page
         * tables have been setup.
         */
        dftss0->tss_eip = (uint32_t)syserrtrap;
        dftss0->tss_esp = (uint32_t)&dblfault_stack0[sizeof (dblfault_stack0)];
        dftss0->tss_cs  = KCS_SEL;
        dftss0->tss_ds  = KDS_SEL;
        dftss0->tss_es  = KDS_SEL;
        dftss0->tss_ss  = KDS_SEL;
        dftss0->tss_fs  = KFS_SEL;
        dftss0->tss_gs  = KGS_SEL;

        /*
         * Set I/O bit map offset equal to size of TSS segment limit
         * for no I/O permission map. This will force all user I/O
         * instructions to generate #gp fault.
         */
        ktss0->tss_bitmapbase = sizeof (*ktss0);

        /*
         * Point %tr to descriptor for ktss0 in gdt.
         */
        wr_tsr(KTSS_SEL);
}

#endif  /* __i386 */
#endif  /* !__xpv */

#if defined(__xpv)
```

```
1196 void
1197 init_desctbls(void)
1198 {
1199         uint_t vec;
1200         user_desc_t *gdt;

1202         /*
1203          * Setup and install our GDT.
1204          */
1205         gdt = init_gdt();

1207         /*
1208          * Store static pa of gdt to speed up pa_to_ma() translations
1209          * on lwp context switches.
1210          */
1211         ASSERT(IS_P2ALIGNED((uintptr_t)gdt, PAGESIZE));
1212         CPU->cpu_gdt = gdt;
1213         CPU->cpu_m.mcpu_gdtpa = pfn_to_pa(va_to_pfn(gdt));

1215         /*
1216          * Setup and install our IDT.
1217          */
1218 #if !defined(__lint)
1219         ASSERT(NIDT * sizeof (*idt0) <= PAGESIZE);
1220 #endif
1221         idt0 = (gate_desc_t *)BOP_ALLOC(bootops, (caddr_t)IDT_VA,
1222             PAGESIZE, PAGESIZE);
1223         bzero(idt0, PAGESIZE);
1224         init_idt(idt0);
1225         for (vec = 0; vec < NIDT; vec++)
1226                 xen_idt_write(&idt0[vec], vec);

1228         CPU->cpu_idt = idt0;

1230         /*
1231          * set default kernel stack
1232          */
1233         xen_stack_switch(KDS_SEL,
1234             (ulong_t)&dblfault_stack0[sizeof (dblfault_stack0)]);

1236         xen_init_callbacks();

1238         init_ldt();
1239 }

1241 #else   /* __xpv */

1243 void
1244 init_desctbls(void)
1245 {
1246         user_desc_t *gdt;
1247         desctbr_t idtr;

1249         /*
1250          * Allocate IDT and TSS structures on unique pages for better
1251          * performance in virtual machines.
1252          */
1253 #if !defined(__lint)
1254         ASSERT(NIDT * sizeof (*idt0) <= PAGESIZE);
1255 #endif
1256         idt0 = (gate_desc_t *)BOP_ALLOC(bootops, (caddr_t)IDT_VA,
1257             PAGESIZE, PAGESIZE);
1258         bzero(idt0, PAGESIZE);
1259 #if !defined(__lint)
1260         ASSERT(sizeof (*ktss0) <= PAGESIZE);
```



```
1261 #endif
1262         ktss0 = (tss_t *)BOP_ALLOC(bootops, (caddr_t)KTSS_VA,
1263             PAGESIZE, PAGESIZE);
1264         bzero(ktss0, PAGESIZE);

1266 #if defined(__i386)
1267 #if !defined(__lint)
1268         ASSERT(sizeof (*dftss0) <= PAGESIZE);
1269 #endif
1270         dftss0 = (tss_t *)BOP_ALLOC(bootops, (caddr_t)DFTSS_VA,
1271             PAGESIZE, PAGESIZE);
1272         bzero(dftss0, PAGESIZE);
1273 #endif

1275         /*
1276          * Setup and install our GDT.
1277          */
1278         gdt = init_gdt();
1279         ASSERT(IS_P2ALIGNED((uintptr_t)gdt, PAGESIZE));
1280         CPU->cpu_gdt = gdt;

1282         /*
1283          * Initialize this CPU's LDT.
1284          */
1285         CPU->cpu_m.mcpu_ldt = BOP_ALLOC(bootops, (caddr_t)LDT_VA,
1286             LDT_CPU_SIZE, PAGESIZE);
1287         bzero(CPU->cpu_m.mcpu_ldt, LDT_CPU_SIZE);
1288         CPU->cpu_m.mcpu_ldt_len = 0;

1290         /*
1291          * Setup and install our IDT.
1292          */
1293         init_idt(idt0);

1295         idtr.dtr_base = (uintptr_t)idt0;
1296         idtr.dtr_limit = (NIDT * sizeof (*idt0)) - 1;
1297         wr_idtr(&idtr);
1298         CPU->cpu_idt = idt0;

1300 #if defined(__i386)
1301         /*
1302          * We maintain a description of idt0 in convenient IDTR format
1303          * for #pf's on some older pentium processors. See pentium_pftrap().
1304          */
1305         idt0_default_r = idtr;
1306 #endif  /* __i386 */

1308         init_tss();
1309         CPU->cpu_tss = ktss0;
1310         init_ldt();

1312         /* Stash this so that the NMI,MCE,#DF and KDI handlers can use it. */
1313         kpti_safe_cr3 = (uint64_t)getcr3();
1314 }
```

_____***unchanged_portion_omitted_***

```
1339 /*
1340  * Enable interpositioning on the system call path by rewriting the
1341  * sys{call|enter} MSRs and the syscall-related entries in the IDT to use
1342  * the branded entry points.
1343  */
1344 void
1345 brand_interpositioning_enable(void)
1346 {
1347         gate_desc_t     *idt = CPU->cpu_idt;
1348         int             i;
```

```
1350         ASSERT(curthread->t_preempt != 0 || getpil() >= DISP_LEVEL);

1352         for (i = 0; brand_tbl[i].ih_inum; i++) {
1353                 idt[brand_tbl[i].ih_inum] = brand_tbl[i].ih_interp_desc;
1354 #if defined(__xpv)
1355                 xen_idt_write(&idt[brand_tbl[i].ih_inum],
1356                     brand_tbl[i].ih_inum);
1357 #endif
1358         }

1360 #if defined(__amd64)
1361 #if defined(__xpv)

1363         /*
1364          * Currently the hypervisor only supports 64-bit syscalls via
1365          * syscall instruction. The 32-bit syscalls are handled by
1366          * interrupt gate above.
1367          */
1368         xen_set_callback(brand_sys_syscall, CALLBACKTYPE_syscall,
1369             CALLBACKF_mask_events);

1371 #else

1373         if (is_x86_feature(x86_featureset, X86FSET_ASYSC)) {
1374                 if (kpti_enable == 1) {
1375                         wrmsr(MSR_AMD_LSTAR, (uintptr_t)tr_brand_sys_syscall);
1376                         wrmsr(MSR_AMD_CSTAR, (uintptr_t)tr_brand_sys_syscall32);
1377                 } else {
1378                         wrmsr(MSR_AMD_LSTAR, (uintptr_t)brand_sys_syscall);
1379                         wrmsr(MSR_AMD_CSTAR, (uintptr_t)brand_sys_syscall32);
1380                 }
1381         }

1383 #endif
1384 #endif  /* __amd64 */

1386         if (is_x86_feature(x86_featureset, X86FSET_SEP)) {
1387                 if (kpti_enable == 1) {
1388                         wrmsr(MSR_INTC_SEP_EIP,
1389                             (uintptr_t)tr_brand_sys_sysenter);
1390                 } else {
1347         if (is_x86_feature(x86_featureset, X86FSET_SEP))
1391                         wrmsr(MSR_INTC_SEP_EIP, (uintptr_t)brand_sys_sysenter);
1392                 }
1393         }
1394 }

1396 /*
1397  * Disable interpositioning on the system call path by rewriting the
1398  * sys{call|enter} MSRs and the syscall-related entries in the IDT to use
1399  * the standard entry points, which bypass the interpositioning hooks.
1400  */
1401 void
1402 brand_interpositioning_disable(void)
1403 {
1404         gate_desc_t     *idt = CPU->cpu_idt;
1405         int i;

1407         ASSERT(curthread->t_preempt != 0 || getpil() >= DISP_LEVEL);

1409         for (i = 0; brand_tbl[i].ih_inum; i++) {
1410                 idt[brand_tbl[i].ih_inum] = brand_tbl[i].ih_default_desc;
1411 #if defined(__xpv)
1412                 xen_idt_write(&idt[brand_tbl[i].ih_inum],
1413                     brand_tbl[i].ih_inum);
```



```
1414 #endif
1415         }

1417 #if defined(__amd64)
1418 #if defined(__xpv)

1420         /*
1421          * See comment above in brand_interpositioning_enable.
1422          */
1423         xen_set_callback(sys_syscall, CALLBACKTYPE_syscall,
1424             CALLBACKF_mask_events);

1426 #else

1428         if (is_x86_feature(x86_featureset, X86FSET_ASYSC)) {
1429                 if (kpti_enable == 1) {
1430                         wrmsr(MSR_AMD_LSTAR, (uintptr_t)tr_sys_syscall);
1431                         wrmsr(MSR_AMD_CSTAR, (uintptr_t)tr_sys_syscall32);
1432                 } else {
1433                         wrmsr(MSR_AMD_LSTAR, (uintptr_t)sys_syscall);
1434                         wrmsr(MSR_AMD_CSTAR, (uintptr_t)sys_syscall32);
1435                 }
1436         }

1438 #endif
1439 #endif  /* __amd64 */

1441         if (is_x86_feature(x86_featureset, X86FSET_SEP)) {
1442                 if (kpti_enable == 1) {
1443                         wrmsr(MSR_INTC_SEP_EIP, (uintptr_t)tr_sys_sysenter);
1444                 } else {
1391         if (is_x86_feature(x86_featureset, X86FSET_SEP))
1445                         wrmsr(MSR_INTC_SEP_EIP, (uintptr_t)sys_sysenter);
1446                 }
1447         }
1448 }
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   21718 Fri Apr  6 17:25:12 2018
new/usr/src/uts/intel/ia32/os/sysi86.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1992, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2018 Joyent, Inc.
 */

/*      Copyright (c) 1990, 1991 UNIX System Laboratories, Inc. */
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989, 1990 AT&T   */
/*        All Rights Reserved   */

/*      Copyright (c) 1987, 1988 Microsoft Corporation  */
/*        All Rights Reserved   */

#include <sys/param.h>
#include <sys/types.h>
#include <sys/sysmacros.h>
#include <sys/systm.h>
#include <sys/signal.h>
#include <sys/errno.h>
#include <sys/fault.h>
#include <sys/syscall.h>
#include <sys/cpuvar.h>
#include <sys/sysi86.h>
#include <sys/psw.h>
#include <sys/cred.h>
#include <sys/policy.h>
#include <sys/thread.h>
#include <sys/debug.h>
#include <sys/ontrap.h>
#include <sys/privregs.h>
#include <sys/x86_archext.h>
#include <sys/vmem.h>
#include <sys/kmem.h>
#include <sys/mman.h>
#include <sys/archsystm.h>
#include <vm/hat.h>
#include <vm/as.h>
#include <vm/seg.h>
#include <vm/seg_kmem.h>
#include <vm/faultcode.h>
```



```
#include <sys/fp.h>
#include <sys/cmn_err.h>
#include <sys/segments.h>
#include <sys/clock.h>
#include <vm/hat_i86.h>
#if defined(__xpv)
#include <sys/hypervisor.h>
#include <sys/note.h>
#endif

static void ldt_alloc(proc_t *, uint_t);
static void ldt_free(proc_t *);
static void ldt_dup(proc_t *, proc_t *);
static void ldt_grow(proc_t *, uint_t);

/*
 * sysi86 System Call
 */

/* ARGSUSED */
int
sysi86(short cmd, uintptr_t arg1, uintptr_t arg2, uintptr_t arg3)
{
        struct ssd ssd;
        int error = 0;
        int c;
        proc_t *pp = curproc;

        switch (cmd) {

        /*
         * The SI86V86 subsystem call of the SYSI86 system call
         * supports only one subcode -- V86SC_IOPL.
         */
        case SI86V86:
                if (arg1 == V86SC_IOPL) {
                        struct regs *rp = lwptoregs(ttolwp(curthread));
                        greg_t oldpl = rp->r_ps & PS_IOPL;
                        greg_t newpl = arg2 & PS_IOPL;

                        /*
                         * Must be privileged to run this system call
                         * if giving more io privilege.
                         */
                        if (newpl > oldpl && (error =
                            secpolicy_sys_config(CRED(), B_FALSE)) != 0)
                                return (set_errno(error));
#if defined(__xpv)
                        kpreempt_disable();
                        installctx(curthread, NULL, xen_disable_user_iopl,
                            xen_enable_user_iopl, NULL, NULL,
                            xen_disable_user_iopl, NULL);
                        xen_enable_user_iopl();
                        kpreempt_enable();
#else
                        rp->r_ps ^= oldpl ^ newpl;
#endif
                } else
                        error = EINVAL;
                break;

        /*
         * Set a segment descriptor
         */
        case SI86DSCR:
                /*
```

```
126                  * There are considerable problems here manipulating
127                  * resources shared by many running lwps.  Get everyone
128                  * into a safe state before changing the LDT.
129                  */
130                 if (curthread != pp->p_agenttp && !holdlwps(SHOLDFORK1)) {
131                         error = EINTR;
132                         break;
133                 }
135                 if (get_udatamodel() == DATAMODEL_LP64) {
136                         error = EINVAL;
137                         break;
138                 }
140                 if (copyin((caddr_t)arg1, &ssd, sizeof (ssd)) < 0) {
141                         error = EFAULT;
142                         break;
143                 }
145                 error = setdscr(&ssd);
147                 mutex_enter(&pp->p_lock);
148                 if (curthread != pp->p_agenttp)
149                         continuelwps(pp);
150                 mutex_exit(&pp->p_lock);
151                 break;
153         case SI86FPHW:
154                 c = fp_kind & 0xff;
155                 if (suword32((void *)arg1, c) == -1)
156                         error = EFAULT;
157                 break;
159         case SI86FPSTART:
160                 /*
161                  * arg1 is the address of _fp_hw
162                  * arg2 is the desired x87 FCW value
163                  * arg3 is the desired SSE MXCSR value
164                  * a return value of one means SSE hardware, else none.
165                  */
166                 c = fp_kind & 0xff;
167                 if (suword32((void *)arg1, c) == -1) {
168                         error = EFAULT;
169                         break;
170                 }
171                 fpsetcw((uint16_t)arg2, (uint32_t)arg3);
172                 return ((fp_kind & __FP_SSE) ? 1 : 0);
174         /* real time clock management commands */
176         case WTODC:
177                 if ((error = secpolicy_settime(CRED())) == 0) {
178                         timestruc_t ts;
179                         mutex_enter(&tod_lock);
180                         gethrestime(&ts);
181                         tod_set(ts);
182                         mutex_exit(&tod_lock);
183                 }
184                 break;
186 /* Give some timezone playing room */
187 #define ONEWEEK (7 * 24 * 60 * 60)
189         case SGMTL:
190                 /*
191                  * Called from 32 bit land, negative values
```



```
192                  * are not sign extended, so we do that here
193                  * by casting it to an int and back.  We also
194                  * clamp the value to within reason and detect
195                  * when a 64 bit call overflows an int.
196                  */
197                 if ((error = secpolicy_settime(CRED())) == 0) {
198                         int newlag = (int)arg1;
200 #ifdef _SYSCALL32_IMPL
201                         if (get_udatamodel() == DATAMODEL_NATIVE &&
202                             (long)newlag != (long)arg1) {
203                                 error = EOVERFLOW;
204                         } else
205 #endif
206                         if (newlag >= -ONEWEEK && newlag <= ONEWEEK)
207                                 sgmtl(newlag);
208                         else
209                                 error = EOVERFLOW;
210                 }
211                 break;
213         case GGMTL:
214                 if (get_udatamodel() == DATAMODEL_NATIVE) {
215                         if (sulword((void *)arg1, ggmtl()) == -1)
216                                 error = EFAULT;
217 #ifdef _SYSCALL32_IMPL
218                 } else {
219                         time_t gmtl;
221                         if ((gmtl = ggmtl()) > INT32_MAX) {
222                                 /*
223                                  * Since gmt_lag can at most be
224                                  * +/- 12 hours, something is
225                                  * *seriously* messed up here.
226                                  */
227                                 error = EOVERFLOW;
228                         } else if (suword32((void *)arg1, (int32_t)gmtl) == -1)
229                                 error = EFAULT;
230 #endif
231                 }
232                 break;
234         case RTCSYNC:
235                 if ((error = secpolicy_settime(CRED())) == 0)
236                         rtcsync();
237                 break;
239         /* END OF real time clock management commands */
241         default:
242                 error = EINVAL;
243                 break;
244         }
245         return (error == 0 ? 0 : set_errno(error));
246 }
______unchanged_portion_omitted_

339 #endif  /* __i386 */
341 /*
342  * Load LDT register with the current process's LDT.
343  */
344 static void
345 ldt_load(void)
346 {
347 #if defined(__xpv)
```

```
348         xen_set_ldt(get_ssd_base(&curproc->p_ldt_desc),
349             curproc->p_ldtlimit + 1);
350 #else
351         size_t len;
352         system_desc_t desc;

354         /*
355          * Before we can use the LDT on this CPU, we must install the LDT in the
356          * user mapping table.
357          */
358         len = (curproc->p_ldtlimit + 1) * sizeof (user_desc_t);
359         bcopy(curproc->p_ldt, CPU->cpu_m.mcpu_ldt, len);
360         CPU->cpu_m.mcpu_ldt_len = len;
361         set_syssegd(&desc, CPU->cpu_m.mcpu_ldt, len - 1, SDT_SYSLDT, SEL_KPL);
362         *((system_desc_t *)&CPU->cpu_gdt[GDT_LDT]) = desc;

349         *((system_desc_t *)&CPU->cpu_gdt[GDT_LDT]) = curproc->p_ldt_desc;
364         wr_ldtr(ULDT_SEL);
365 #endif
366 }

368 /*
369  * Store a NULL selector in the LDTR. All subsequent illegal references to
370  * the LDT will result in a #gp.
371  */
372 void
373 ldt_unload(void)
374 {
375 #if defined(__xpv)
376         xen_set_ldt(NULL, 0);
377 #else
378         *((system_desc_t *)&CPU->cpu_gdt[GDT_LDT]) = null_sdesc;
379         wr_ldtr(0);

381         bzero(CPU->cpu_m.mcpu_ldt, CPU->cpu_m.mcpu_ldt_len);
382         CPU->cpu_m.mcpu_ldt_len = 0;
383 #endif
384 }
```

unchanged_portion_omitted_

```
715 /*
716  * Allocate new LDT for process just large enough to contain seli.
717  * Note we allocate and grow LDT in PAGESIZE chunks. We do this
718  * to simplify the implementation and because on the hypervisor it's
719  * required, since the LDT must live on pages that have PROT_WRITE
720  * removed and which are given to the hypervisor.
721  */
722 static void
723 ldt_alloc(proc_t *pp, uint_t seli)
724 {
725         user_desc_t     *ldt;
726         size_t          ldtsz;
727         uint_t          nsels;

729         ASSERT(MUTEX_HELD(&pp->p_ldtlock));
730         ASSERT(pp->p_ldt == NULL);
731         ASSERT(pp->p_ldtlimit == 0);

733         /*
734          * Allocate new LDT just large enough to contain seli. The LDT must
735          * always be allocated in units of pages for KPTI.
717          * Allocate new LDT just large enough to contain seli.
736          */
737         ldtsz = P2ROUNDUP((seli + 1) * sizeof (user_desc_t), PAGESIZE);
738         nsels = ldtsz / sizeof (user_desc_t);
739         ASSERT(nsels >= MINNLDT && nsels <= MAXNLDT);
```



```
741         ldt = kmem_zalloc(ldtsz, KM_SLEEP);
742         ASSERT(IS_P2ALIGNED(ldt, PAGESIZE));

744 #if defined(__xpv)
745         if (xen_ldt_setprot(ldt, ldtsz, PROT_READ))
746                 panic("ldt_alloc:xen_ldt_setprot(PROT_READ) failed");
747 #endif

749         pp->p_ldt = ldt;
750         pp->p_ldtlimit = nsels - 1;
751         set_syssegd(&pp->p_ldt_desc, ldt, ldtsz - 1, SDT_SYSLDT, SEL_KPL);

753         if (pp == curproc) {
754                 kpreempt_disable();
755                 ldt_load();
756                 kpreempt_enable();
757         }
758 }
```

unchanged_portion_omitted_

```
841 static void
842 ldt_grow(proc_t *pp, uint_t seli)
843 {
844         user_desc_t     *oldt, *nldt;
845         uint_t          nsels;
846         size_t          oldtsz, nldtsz;

848         ASSERT(MUTEX_HELD(&pp->p_ldtlock));
849         ASSERT(pp->p_ldt != NULL);
850         ASSERT(pp->p_ldtlimit != 0);

852         /*
853          * Allocate larger LDT just large enough to contain seli. The LDT must
854          * always be allocated in units of pages for KPTI.
835          * Allocate larger LDT just large enough to contain seli.
855          */
856         nldtsz = P2ROUNDUP((seli + 1) * sizeof (user_desc_t), PAGESIZE);
857         nsels = nldtsz / sizeof (user_desc_t);
858         ASSERT(nsels >= MINNLDT && nsels <= MAXNLDT);
859         ASSERT(nsels > pp->p_ldtlimit);

861         oldt = pp->p_ldt;
862         oldtsz = (pp->p_ldtlimit + 1) * sizeof (user_desc_t);

864         nldt = kmem_zalloc(nldtsz, KM_SLEEP);
865         ASSERT(IS_P2ALIGNED(nldt, PAGESIZE));

867         bcopy(oldt, nldt, oldtsz);

869         /*
870          * unload old ldt.
871          */
872         kpreempt_disable();
873         ldt_unload();
874         kpreempt_enable();

876 #if defined(__xpv)

878         /*
879          * Make old ldt writable and new ldt read only.
880          */
881         if (xen_ldt_setprot(oldt, oldtsz, PROT_READ | PROT_WRITE))
882                 panic("ldt_grow:xen_ldt_setprot(PROT_READ|PROT_WRITE) failed");

884         if (xen_ldt_setprot(nldt, nldtsz, PROT_READ))
```

```
885                 panic("ldt_grow:xen_ldt_setprot(PROT_READ) failed");
886 #endif

888         pp->p_ldt = nldt;
889         pp->p_ldtlimit = nsels - 1;

891         /*
892          * write new ldt segment descriptor.
893          */
894         set_syssegd(&pp->p_ldt_desc, nldt, nldtsz - 1, SDT_SYSLDT, SEL_KPL);

896         /*
897          * load the new ldt.
898          */
899         kpreempt_disable();
900         ldt_load();
901         kpreempt_enable();

903         kmem_free(oldt, oldtsz);
904 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    1850 Fri Apr  6 17:25:12 2018
new/usr/src/uts/intel/ia32/sys/kdi_regs.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 _IA32_SYS_KDI_REGS_H
#define _IA32_SYS_KDI_REGS_H

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

#ifdef __cplusplus
extern "C" {
#endif

#define KDIREG_NGREG    21

/*
 * %ss appears in a different place than a typical struct regs, since the
 * machine won't save %ss on a trap entry from the same privilege level.
 */

#define KDIREG_SAVFP    0
#define KDIREG_SAVPC    1
#define KDIREG_SS       2
#define KDIREG_GS       3
#define KDIREG_FS       4
#define KDIREG_ES       5
#define KDIREG_DS       6
#define KDIREG_EDI      7
#define KDIREG_ESI      8
#define KDIREG_EBP      9
#define KDIREG_ESP      10
#define KDIREG_EBX      11
#define KDIREG_EDX      12
```



```
#define KDIREG_ECX      13
#define KDIREG_EAX      14
#define KDIREG_TRAPNO   15
#define KDIREG_ERR      16
#define KDIREG_EIP      17
#define KDIREG_CS       18
#define KDIREG_EFLAGS   19
#define KDIREG_UESP     20

#define KDIREG_PC       KDIREG_EIP
#define KDIREG_SP       KDIREG_ESP
#define KDIREG_FP       KDIREG_EBP

#ifdef _ASM

/* Patch point for MSR clearing. */
#define KDI_MSR_PATCH \
        nop; nop; nop; nop; \
        nop; nop; nop; nop; \
        nop; nop; nop; nop; \
        nop

#endif /* _ASM */

#define KDI_MSR_PATCHOFF        8       /* bytes of code before patch point */
#define KDI_MSR_PATCHSZ         13      /* bytes in KDI_MSR_PATCH, above */

#ifdef __cplusplus
}
```

```
______unchanged_portion_omitted_
```

```
**********************************************************
    4210 Fri Apr  6 17:25:12 2018
new/usr/src/uts/intel/ia32/sys/trap.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1990, 1991 UNIX System Laboratories, Inc. */
/*      Copyright (c) 1984, 1986, 1987, 1988, 1989, 1990 AT&T   */
/*        All Rights Reserved   */

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

#ifndef _IA32_SYS_TRAP_H
#define _IA32_SYS_TRAP_H

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Trap type values
 */

#define T_ZERODIV       0x0     /* #de  divide by 0 error               */
#define T_SGLSTP        0x1     /* #db  single step                     */
#define T_NMIFLT        0x2     /*      NMI                             */
#define T_BPTFLT        0x3     /* #bp  breakpoint fault, INT3 insn     */
#define T_OVFLW         0x4     /* #of  INTO overflow fault             */
#define T_BOUNDFLT      0x5     /* #br  BOUND insn fault                */
#define T_ILLINST       0x6     /* #ud  invalid opcode fault            */
#define T_NOEXTFLT      0x7     /* #nm  device not available: x87       */
#define T_DBLFLT        0x8     /* #df  double fault                    */
#define T_EXTOVRFLT     0x9     /*      [not generated: 386 only]       */
#define T_TSSFLT        0xa     /* #ts  invalid TSS fault               */
#define T_SEGFLT        0xb     /* #np  segment not present fault       */
#define T_STKFLT        0xc     /* #ss  stack fault                     */
#define T_GPFLT         0xd     /* #gp  general protection fault        */
#define T_PGFLT         0xe     /* #pf  page fault                      */
#define T_RESVTRAP      0xf     /*      reserved                        */
#define T_EXTERRFLT     0x10    /* #mf  x87 FPU error fault             */
```



```
#define T_ALIGNMENT     0x11    /* #ac  alignment check error           */
#define T_MCE           0x12    /* #mc  machine check exception         */
#define T_SIMDFPE       0x13    /* #xm  SSE/SSE exception               */
#define T_DBGENTR       0x14    /*      debugger entry                  */
#define T_INVALTRAP     0x1e    /*      invalid                         */
#define T_ENDPERR       0x21    /*      emulated extension error flt    */
#define T_ENOEXTFLT     0x20    /*      emulated ext not present        */
#define T_FASTTRAP      0xd2    /*      fast system call                */
#define T_SYSCALLINT    0x91    /*      general system call             */
#define T_DTRACE_RET    0x92    /*      DTrace pid return               */
#define T_INT80         0x80    /*      int80 handler for linux emulation */
#define T_SOFTINT       0x50fd  /*      pseudo softint trap type        */

/*
 * Pseudo traps.
 */
#define T_INTERRUPT             0x100
#define T_FAULT                 0x200
#define T_AST                   0x400
#define T_SYSCALL               0x180


/*
 *  Values of error code on stack in case of page fault
 */

#define PF_ERR_MASK     0x01    /* Mask for error bit */
#define PF_ERR_PAGE     0x00    /* page not present */
#define PF_ERR_PROT     0x01    /* protection error */
#define PF_ERR_WRITE    0x02    /* fault caused by write (else read) */
#define PF_ERR_USER     0x04    /* processor was in user mode */
                                /*      (else supervisor) */
#define PF_ERR_EXEC     0x10    /* attempt to execute a No eXec page (AMD) */
                                /* or kernel tried to execute a user page  */
                                /* (Intel SMEP) */

/*
 *  Definitions for fast system call subfunctions
 */
#define T_FNULL         0       /* Null trap for testing                */
#define T_FGETFP        1       /* Get emulated FP context              */
#define T_FSETFP        2       /* Set emulated FP context              */
#define T_GETHRTIME     3       /* Get high resolution time             */
#define T_GETHRVTIME    4       /* Get high resolution virtual time     */
#define T_GETHRESTIME   5       /* Get high resolution time             */
#define T_GETLGRP       6       /* Get home lgrpid                      */

#define T_LASTFAST      6       /* Last valid subfunction               */

/*
 * Offsets for an interrupt/trap frame.
 */
#define T_FRAME_ERR     0
#define T_FRAME_RIP     8
#define T_FRAME_CS      16
#define T_FRAME_RFLAGS  24
#define T_FRAME_RSP     32
#define T_FRAME_SS      40

#define T_FRAMERET_RIP          0
#define T_FRAMERET_CS           8
#define T_FRAMERET_RFLAGS       16
#define T_FRAMERET_RSP          24
#define T_FRAMERET_SS           32

#ifdef  __cplusplus
```

```
 126 }
______unchanged_portion_omitted_
```

```
**********************************************************
   17745 Fri Apr  6 17:25:12 2018
new/usr/src/uts/intel/kdi/kdi_asm.s
9441 kmdb should stash %cr3 in kdiregs
Reviewed by: John Levon <john.levon@joyent.com>
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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"

/*
 * Debugger entry and exit for both master and slave CPUs. kdi_idthdl.s contains
 * the IDT stubs that drop into here (mainly via kdi_cmnint).
 * Debugger entry for both master and slave CPUs
 */

#if defined(__lint)
#include <sys/types.h>
#else
#endif

#include <sys/segments.h>
#include <sys/asm_linkage.h>
#include <sys/controlregs.h>
#include <sys/x86_archext.h>
#include <sys/privregs.h>
#include <sys/machprivregs.h>
#include <sys/kdi_regs.h>
#include <sys/psw.h>
#include <sys/uadmin.h>
#ifdef __xpv
#include <sys/hypervisor.h>
#endif

#ifdef _ASM

#include <kdi_assym.h>
```



```
#include <assym.h>

/* clobbers %rdx, %rcx, returns addr in %rax, CPU ID in %rbx */
#define GET_CPUSAVE_ADDR \
        movzbq  %gs:CPU_ID, %rbx;               \
        movq    %rbx, %rax;                     \
        movq    $KRS_SIZE, %rcx;                \
        mulq    %rcx;                           \
        movq    $kdi_cpusave, %rdx;             \
        /*CSTYLED*/                             \
        addq    (%rdx), %rax

/*
 * Save copies of the IDT and GDT descriptors.  Note that we only save the IDT
 * and GDT if the IDT isn't ours, as we may be legitimately re-entering the
 * debugger through the trap handler.  We don't want to clobber the saved IDT
 * in the process, as we'd end up resuming the world on our IDT.
 */
#define SAVE_IDTGDT                             \
        movq    %gs:CPU_IDT, %r11;              \
        leaq    kdi_idt(%rip), %rsi;            \
        cmpq    %rsi, %r11;                     \
        je      1f;                             \
        movq    %r11, KRS_IDT(%rax);            \
        movq    %gs:CPU_GDT, %r11;              \
        movq    %r11, KRS_GDT(%rax);            \
1:

#ifdef __xpv

/*
 * Already on kernel gsbase via the hypervisor.
 */
#define SAVE_GSBASE(reg) /* nothing */
#define RESTORE_GSBASE(reg) /* nothing */

#else

#define SAVE_GSBASE(base)                               \
        movl    $MSR_AMD_GSBASE, %ecx;                  \
        rdmsr;                                          \
        shlq    $32, %rdx;                              \
        orq     %rax, %rdx;                             \
        movq    %rdx, REG_OFF(KDIREG_GSBASE)(base);     \
        movl    $MSR_AMD_KGSBASE, %ecx;                 \
        rdmsr;                                          \
        shlq    $32, %rdx;                              \
        orq     %rax, %rdx;                             \
        movq    %rdx, REG_OFF(KDIREG_KGSBASE)(base)
        movq    %rdx, REG_OFF(KDIREG_GSBASE)(base)

/*
 * We shouldn't have stomped on KGSBASE, so don't try to restore it.
 */
#define RESTORE_GSBASE(base)                            \
        movq    REG_OFF(KDIREG_GSBASE)(base), %rdx;     \
        movq    %rdx, %rax;                             \
        shrq    $32, %rdx;                              \
        movl    $MSR_AMD_GSBASE, %ecx;                  \
        wrmsr

#endif /* __xpv */

/*
 * %ss, %rsp, %rflags, %cs, %rip, %err, %trapno are already on the stack.
 * %ss, %rsp, %rflags, %cs, %rip, %err, %trapno are already on the stack.  Note
```

```
106  * that on the hypervisor, we skip the save/restore of GSBASE: it's slow, and
107  * unnecessary.
115  */
116 #define KDI_SAVE_REGS(base) \
117         movq    %rdi, REG_OFF(KDIREG_RDI)(base);        \
118         movq    %rsi, REG_OFF(KDIREG_RSI)(base);        \
119         movq    %rdx, REG_OFF(KDIREG_RDX)(base);        \
120         movq    %rcx, REG_OFF(KDIREG_RCX)(base);        \
121         movq    %r8, REG_OFF(KDIREG_R8)(base);          \
122         movq    %r9, REG_OFF(KDIREG_R9)(base);          \
123         movq    %rax, REG_OFF(KDIREG_RAX)(base);        \
124         movq    %rbx, REG_OFF(KDIREG_RBX)(base);        \
125         movq    %rbp, REG_OFF(KDIREG_RBP)(base);        \
126         movq    %r10, REG_OFF(KDIREG_R10)(base);        \
127         movq    %r11, REG_OFF(KDIREG_R11)(base);        \
128         movq    %r12, REG_OFF(KDIREG_R12)(base);        \
129         movq    %r13, REG_OFF(KDIREG_R13)(base);        \
130         movq    %r14, REG_OFF(KDIREG_R14)(base);        \
131         movq    %r15, REG_OFF(KDIREG_R15)(base);        \
132         movq    %rbp, REG_OFF(KDIREG_SAVFP)(base);      \
133         movq    REG_OFF(KDIREG_RIP)(base), %rax;        \
134         movq    %rax, REG_OFF(KDIREG_SAVPC)(base);      \
135         movq    %cr2, %rax;                             \
136         movq    %rax, REG_OFF(KDIREG_CR2)(base);        \
137         clrq    %rax;                                   \
138         movw    %ds, %ax;                               \
139         movq    %rax, REG_OFF(KDIREG_DS)(base);         \
140         movw    %es, %ax;                               \
141         movq    %rax, REG_OFF(KDIREG_ES)(base);         \
142         movw    %fs, %ax;                               \
143         movq    %rax, REG_OFF(KDIREG_FS)(base);         \
144         movw    %gs, %ax;                               \
145         movq    %rax, REG_OFF(KDIREG_GS)(base);         \
146         SAVE_GSBASE(base)

148 #define KDI_RESTORE_REGS(base) \
149         movq    base, %rdi;                             \
150         RESTORE_GSBASE(%rdi);                           \
151         movq    REG_OFF(KDIREG_ES)(%rdi), %rax;         \
152         movw    %ax, %es;                               \
153         movq    REG_OFF(KDIREG_DS)(%rdi), %rax;         \
154         movw    %ax, %ds;                               \
155         movq    REG_OFF(KDIREG_CR2)(base), %rax;        \
156         movq    %rax, %cr2;                             \
157         movq    REG_OFF(KDIREG_R15)(%rdi), %r15;        \
158         movq    REG_OFF(KDIREG_R14)(%rdi), %r14;        \
159         movq    REG_OFF(KDIREG_R13)(%rdi), %r13;        \
160         movq    REG_OFF(KDIREG_R12)(%rdi), %r12;        \
161         movq    REG_OFF(KDIREG_R11)(%rdi), %r11;        \
162         movq    REG_OFF(KDIREG_R10)(%rdi), %r10;        \
163         movq    REG_OFF(KDIREG_RBP)(%rdi), %rbp;        \
164         movq    REG_OFF(KDIREG_RBX)(%rdi), %rbx;        \
165         movq    REG_OFF(KDIREG_RAX)(%rdi), %rax;        \
166         movq    REG_OFF(KDIREG_R9)(%rdi), %r9;          \
167         movq    REG_OFF(KDIREG_R8)(%rdi), %r8;          \
168         movq    REG_OFF(KDIREG_RCX)(%rdi), %rcx;        \
169         movq    REG_OFF(KDIREG_RDX)(%rdi), %rdx;        \
170         movq    REG_OFF(KDIREG_RSI)(%rdi), %rsi;        \
171         movq    REG_OFF(KDIREG_RDI)(%rdi), %rdi

173 /*
174  * Given the address of the current CPU's cpusave area in %rax, the following
175  * macro restores the debugging state to said CPU.  Restored state includes
176  * the debug registers from the global %dr variables.
165  * the debug registers from the global %dr variables, and debugging MSRs from
166  * the CPU save area.  This code would be in a separate routine, but for the
```



```
167  * fact that some of the MSRs are jump-sensitive.  As such, we need to minimize
168  * the number of jumps taken subsequent to the update of said MSRs.  We can
169  * remove one jump (the ret) by using a macro instead of a function for the
170  * debugging state restoration code.
177  *
178  * Takes the cpusave area in %rdi as a parameter.
172  * Takes the cpusave area in %rdi as a parameter, clobbers %rax-%rdx
179  */
180 #define KDI_RESTORE_DEBUGGING_STATE \
181         pushq   %rdi;                                                   \
182         leaq    kdi_drreg(%rip), %r15;                                  \
183         movl    $7, %edi;                                               \
184         movq    DR_CTL(%r15), %rsi;                                     \
185         call    kdi_dreg_set;                                           \
186                                                                         \
187         movl    $6, %edi;                                               \
188         movq    $KDIREG_DRSTAT_RESERVED, %rsi;                          \
189         call    kdi_dreg_set;                                           \
190                                                                         \
191         movl    $0, %edi;                                               \
192         movq    DRADDR_OFF(0)(%r15), %rsi;                              \
193         call    kdi_dreg_set;                                           \
194         movl    $1, %edi;                                               \
195         movq    DRADDR_OFF(1)(%r15), %rsi;                              \
196         call    kdi_dreg_set;                                           \
197         movl    $2, %edi;                                               \
198         movq    DRADDR_OFF(2)(%r15), %rsi;                              \
199         call    kdi_dreg_set;                                           \
200         movl    $3, %edi;                                               \
201         movq    DRADDR_OFF(3)(%r15), %rsi;                              \
202         call    kdi_dreg_set;                                           \
203         popq    %rdi;
197         popq    %rdi;                                                   \
198                                                                         \
199         /*                                                              \
200          * Write any requested MSRs.                                    \
201          */                                                             \
202         movq    KRS_MSR(%rdi), %rbx;                                    \
203         cmpq    $0, %rbx;                                               \
204         je      3f;                                                     \
205 1:                                                                      \
206         movl    MSR_NUM(%rbx), %ecx;                                    \
207         cmpl    $0, %ecx;                                               \
208         je      3f;                                                     \
209                                                                         \
210         movl    MSR_TYPE(%rbx), %edx;                                   \
211         cmpl    $KDI_MSR_WRITE, %edx;                                   \
212         jne     2f;                                                     \
213                                                                         \
214         movq    MSR_VALP(%rbx), %rdx;                                   \
215         movl    0(%rdx), %eax;                                          \
216         movl    4(%rdx), %edx;                                          \
217         wrmsr;                                                          \
218 2:                                                                      \
219         addq    $MSR_SIZE, %rbx;                                        \
220         jmp     1b;                                                     \
221 3:                                                                      \
222         /*                                                              \
223          * We must not branch after re-enabling LBR.  If               \
224          * kdi_wsr_wrexit_msr is set, it contains the number           \
225          * of the MSR that controls LBR.  kdi_wsr_wrexit_valp          \
226          * contains the value that is to be written to enable          \
227          * LBR.                                                        \
228          */                                                             \
229         leaq    kdi_msr_wrexit_msr(%rip), %rcx;                         \
230         movl    (%rcx), %ecx;                                           \
```

```
231         cmpl    $0, %ecx;                                       \
232         je      1f;                                             \
233                                                                 \
234         leaq    kdi_msr_wrexit_valp(%rip), %rdx;                \
235         movq    (%rdx), %rdx;                                   \
236         movl    0(%rdx), %eax;                                  \
237         movl    4(%rdx), %edx;                                  \
238                                                                 \
239         wrmsr;                                                  \
240 1:

205 /*
206  * Each cpusave buffer has an area set aside for a ring buffer of breadcrumbs.
207  * The following macros manage the buffer.
208  */

210 /* Advance the ring buffer */
211 #define ADVANCE_CRUMB_POINTER(cpusave, tmp1, tmp2) \
212         movq    KRS_CURCRUMBIDX(cpusave), tmp1; \
213         cmpq    $[KDI_NCRUMBS - 1], tmp1;       \
214         jge     1f;                             \
215         /* Advance the pointer and index */     \
216         addq    $1, tmp1;                       \
217         movq    tmp1, KRS_CURCRUMBIDX(cpusave); \
218         movq    KRS_CURCRUMB(cpusave), tmp1;    \
219         addq    $KRM_SIZE, tmp1;                \
220         jmp     2f;                             \
221 1:      /* Reset the pointer and index */       \
222         movq    $0, KRS_CURCRUMBIDX(cpusave);   \
223         leaq    KRS_CRUMBS(cpusave), tmp1;      \
224 2:      movq    tmp1, KRS_CURCRUMB(cpusave);    \
225         /* Clear the new crumb */               \
226         movq    $KDI_NCRUMBS, tmp2;             \
227 3:      movq    $0, -4(tmp1, tmp2, 4);          \
228         decq    tmp2;                           \
229         jnz     3b

231 /* Set a value in the current breadcrumb buffer */
232 #define ADD_CRUMB(cpusave, offset, value, tmp) \
233         movq    KRS_CURCRUMB(cpusave), tmp;     \
234         movq    value, offset(tmp)

273 #endif  /* _ASM */

275 #if defined(__lint)
276 void
277 kdi_cmnint(void)
278 {
279 }
280 #else   /* __lint */

236         /* XXX implement me */
237         ENTRY_NP(kdi_nmiint)
238         clrq    %rcx
239         movq    (%rcx), %rcx
240         SET_SIZE(kdi_nmiint)

242         /*
243          * The main entry point for master CPUs.  It also serves as the trap
244          * handler for all traps and interrupts taken during single-step.
245          */
246         ENTRY_NP(kdi_cmnint)
247         ALTENTRY(kdi_master_entry)

249         pushq   %rax
250         CLI(%rax)
```



```
251         popq    %rax

253         /* Save current register state */
254         subq    $REG_OFF(KDIREG_TRAPNO), %rsp
255         KDI_SAVE_REGS(%rsp)

257 #ifdef __xpv
258         /*
259          * Clear saved_upcall_mask in unused byte of cs slot on stack.
260          * It can only confuse things.
261          */
262         movb    $0, REG_OFF(KDIREG_CS)+4(%rsp)
263 #endif

265 #if !defined(__xpv)
266         /*
267          * Switch to the kernel's GSBASE.  Neither GSBASE nor the ill-named
268          * KGSBASE can be trusted, as the kernel may or may not have already
269          * done a swapgs.  All is not lost, as the kernel can divine the correct
270          * value for us.  Note that the previous GSBASE is saved in the
271          * KDI_SAVE_REGS macro to prevent a usermode process's GSBASE from being
272          * blown away.  On the hypervisor, we don't need to do this, since it's
273          * ensured we're on our requested kernel GSBASE already.
274          */
275         subq    $10, %rsp
276         sgdt    (%rsp)
277         movq    2(%rsp), %rdi   /* gdt base now in %rdi */
278         addq    $10, %rsp
279         call    kdi_gdt2gsbase  /* returns kernel's GSBASE in %rax */

281         movq    %rax, %rdx
282         shrq    $32, %rdx
283         movl    $MSR_AMD_GSBASE, %ecx
284         wrmsr

286         /*
287          * In the trampoline we stashed the incoming %cr3. Copy this into
288          * the kdiregs for restoration and later use.
289          */
290         mov     %gs:(CPU_KPTI_DBG+KPTI_TR_CR3), %rdx
291         mov     %rdx, REG_OFF(KDIREG_CR3)(%rsp)
292         /*
293          * Switch to the kernel's %cr3. From the early interrupt handler
294          * until now we've been running on the "paranoid" %cr3 (that of kas
295          * from early in boot).
296          *
297          * If we took the interrupt from somewhere already on the kas/paranoid
298          * %cr3 though, don't change it (this could happen if kcr3 is corrupt
299          * and we took a gptrap earlier from this very code).
300          */
301         cmpq    %rdx, kpti_safe_cr3
302         je      .no_kcr3
303         mov     %gs:CPU_KPTI_KCR3, %rdx
304         cmpq    $0, %rdx
305         je      .no_kcr3
306         mov     %rdx, %cr3
307 .no_kcr3:

309 #endif  /* __xpv */

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

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

315         ADD_CRUMB(%rax, KRM_CPU_STATE, $KDI_CPU_STATE_MASTER, %rdx)
```

```
317         movq    REG_OFF(KDIREG_RIP)(%rsp), %rcx
318         ADD_CRUMB(%rax, KRM_PC, %rcx, %rdx)
319         ADD_CRUMB(%rax, KRM_SP, %rsp, %rdx)
320         movq    REG_OFF(KDIREG_TRAPNO)(%rsp), %rcx
321         ADD_CRUMB(%rax, KRM_TRAPNO, %rcx, %rdx)

323         movq    %rsp, %rbp
324         pushq   %rax

326         /*
327          * Were we in the debugger when we took the trap (i.e. was %esp in one
328          * of the debugger's memory ranges)?
329          */
330         leaq    kdi_memranges, %rcx
331         movl    kdi_nmemranges, %edx
332 1:
333         cmpq    MR_BASE(%rcx), %rsp
354 1:      cmpq    MR_BASE(%rcx), %rsp
334         jl      2f              /* below this range -- try the next one */
335         cmpq    MR_LIM(%rcx), %rsp
336         jg      2f              /* above this range -- try the next one */
337         jmp     3f              /* matched within this range */

339 2:
340         decl    %edx
360 2:      decl    %edx
341         jz      kdi_save_common_state   /* %rsp not within debugger memory */
342         addq    $MR_SIZE, %rcx
343         jmp     1b

345 3:      /*
346          * The master is still set.  That should only happen if we hit a trap
347          * while running in the debugger.  Note that it may be an intentional
348          * fault.  kmdb_dpi_handle_fault will sort it all out.
349          */

351         movq    REG_OFF(KDIREG_TRAPNO)(%rbp), %rdi
352         movq    REG_OFF(KDIREG_RIP)(%rbp), %rsi
353         movq    REG_OFF(KDIREG_RSP)(%rbp), %rdx
354         movq    %rbx, %rcx              /* cpuid */

356         call    kdi_dvec_handle_fault

358         /*
359          * If we're here, we ran into a debugger problem, and the user
360          * elected to solve it by having the debugger debug itself.  The
361          * state we're about to save is that of the debugger when it took
362          * the fault.
363          */

365         jmp     kdi_save_common_state

367         SET_SIZE(kdi_master_entry)
_____unchanged_portion_omitted_

390 #endif  /* __lint */

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  */
```



```
402 #if defined(__lint)
403 char kdi_slave_entry_patch;

405 void
406 kdi_slave_entry(void)
407 {
408 }
409 #else /* __lint */
410         .globl  kdi_slave_entry_patch;

380         ENTRY_NP(kdi_slave_entry)

414         /* kdi_msr_add_clrentry knows where this is */
415 kdi_slave_entry_patch:
416         KDI_MSR_PATCH;

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         xchgq   REG_OFF(KDIREG_RIP)(%rsp), %rax
410         movq    %rax, REG_OFF(KDIREG_SS)(%rsp)

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

416         /*
417          * We've saved all of the general-purpose registers, and have a stack
418          * that is irettable (after we strip down to the error code)
419          */

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

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

425         ADD_CRUMB(%rax, KRM_CPU_STATE, $KDI_CPU_STATE_SLAVE, %rdx)

427         movq    REG_OFF(KDIREG_RIP)(%rsp), %rcx
428         ADD_CRUMB(%rax, KRM_PC, %rcx, %rdx)

430         pushq   %rax
```

```
431         jmp     kdi_save_common_state

433         SET_SIZE(kdi_slave_entry)

468 #endif  /* __lint */

435 /*
436  * The state of the world:
437  *
438  * The stack has a complete set of saved registers and segment
439  * selectors, arranged in the kdi_regs.h order.  It also has a pointer
440  * to our cpusave area.
441  *
442  * We need to save, into the cpusave area, a pointer to these saved
443  * registers.  First we check whether we should jump straight back to
444  * the kernel.  If not, we save a few more registers, ready the
445  * machine for debugger entry, and enter the debugger.
446  */

483 #if !defined(__lint)

448         ENTRY_NP(kdi_save_common_state)

450         popq    %rdi                    /* the cpusave area */
451         movq    %rsp, KRS_GREGS(%rdi)   /* save ptr to current saved regs */

453         pushq   %rdi
454         call    kdi_trap_pass
455         cmpq    $1, %rax
456         je      kdi_pass_to_kernel
457         popq    %rax /* cpusave in %rax */

459         SAVE_IDTGDT

461 #if !defined(__xpv)
462         /* Save off %cr0, and clear write protect */
463         movq    %cr0, %rcx
464         movq    %rcx, KRS_CR0(%rax)
465         andq    $_BITNOT(CR0_WP), %rcx
466         movq    %rcx, %cr0
467 #endif

469         /* Save the debug registers and disable any active watchpoints */

471         movq    %rax, %r15              /* save cpusave area ptr */
472         movl    $7, %edi
473         call    kdi_dreg_get
474         movq    %rax, KRS_DRCTL(%r15)

476         andq    $_BITNOT(KDIREG_DRCTL_WPALLEN_MASK), %rax
477         movq    %rax, %rsi
478         movl    $7, %edi
479         call    kdi_dreg_set

481         movl    $6, %edi
482         call    kdi_dreg_get
483         movq    %rax, KRS_DRSTAT(%r15)

485         movl    $0, %edi
486         call    kdi_dreg_get
487         movq    %rax, KRS_DROFF(0)(%r15)

489         movl    $1, %edi
490         call    kdi_dreg_get
491         movq    %rax, KRS_DROFF(1)(%r15)
```



```
493         movl    $2, %edi
494         call    kdi_dreg_get
495         movq    %rax, KRS_DROFF(2)(%r15)

497         movl    $3, %edi
498         call    kdi_dreg_get
499         movq    %rax, KRS_DROFF(3)(%r15)

501         movq    %r15, %rax      /* restore cpu save area to rax */

540         /*
541          * Save any requested MSRs.
542          */
543         movq    KRS_MSR(%rax), %rcx
544         cmpq    $0, %rcx
545         je      no_msr

547         pushq   %rax            /* rdmsr clobbers %eax */
548         movq    %rcx, %rbx

550 1:
551         movl    MSR_NUM(%rbx), %ecx
552         cmpl    $0, %ecx
553         je      msr_done

555         movl    MSR_TYPE(%rbx), %edx
556         cmpl    $KDI_MSR_READ, %edx
557         jne     msr_next

559         rdmsr                   /* addr in %ecx, value into %edx:%eax */
560         movl    %eax, MSR_VAL(%rbx)
561         movl    %edx, _CONST(MSR_VAL + 4)(%rbx)

563 msr_next:
564         addq    $MSR_SIZE, %rbx
565         jmp     1b

567 msr_done:
568         popq    %rax

570 no_msr:
503         clrq    %rbp            /* stack traces should end here */

505         pushq   %rax
506         movq    %rax, %rdi      /* cpusave */

508         call    kdi_debugger_entry

510         /* Pass cpusave to kdi_resume */
511         popq    %rdi

513         jmp     kdi_resume

515         SET_SIZE(kdi_save_common_state)

585 #endif  /* !__lint */

517 /*
518  * Resume the world.  The code that calls kdi_resume has already
519  * decided whether or not to restore the IDT.
520  */
591 #if defined(__lint)
592 void
593 kdi_resume(void)
594 {
595 }
```

```
596 #else   /* __lint */

521         /* cpusave in %rdi */
522         ENTRY_NP(kdi_resume)

524         /*
525          * Send this CPU back into the world
526          */
527 #if !defined(__xpv)
528         movq    KRS_CR0(%rdi), %rdx
529         movq    %rdx, %cr0
530 #endif

532         KDI_RESTORE_DEBUGGING_STATE

534         movq    KRS_GREGS(%rdi), %rsp

536 #if !defined(__xpv)
537         /*
538          * If we're going back via tr_iret_kdi, then we want to copy the
539          * final %cr3 we're going to back into the kpti_dbg area now.
540          *
541          * Since the trampoline needs to find the kpti_dbg too, we enter it
542          * with %r13 set to point at that. The real %r13 (to restore before
543          * the iret) we stash in the kpti_dbg itself.
544          */
545         movq    %gs:CPU_SELF, %r13      /* can't leaq %gs:*, use self-ptr */
546         addq    $CPU_KPTI_DBG, %r13

548         movq    REG_OFF(KDIREG_R13)(%rsp), %rdx
549         movq    %rdx, KPTI_R13(%r13)

551         movq    REG_OFF(KDIREG_CR3)(%rsp), %rdx
552         movq    %rdx, KPTI_TR_CR3(%r13)

554         /* The trampoline will undo this later. */
555         movq    %r13, REG_OFF(KDIREG_R13)(%rsp)
556 #endif

558         KDI_RESTORE_REGS(%rsp)
559         addq    $REG_OFF(KDIREG_RIP), %rsp      /* Discard state, trapno, err */
560         /*
561          * The common trampoline code will restore %cr3 to the right value
562          * for either kernel or userland.
563          */
564 #if !defined(__xpv)
565         jmp     tr_iret_kdi
566 #else
567         IRET
568 #endif
569         /*NOTREACHED*/
570         SET_SIZE(kdi_resume)

618 #endif  /* __lint */

620 #if !defined(__lint)

572         ENTRY_NP(kdi_pass_to_kernel)

574         popq    %rdi /* cpusave */

576         movq    $KDI_CPU_STATE_NONE, KRS_CPU_STATE(%rdi)

578         /*
579          * Find the trap and vector off the right kernel handler.  The trap
580          * handler will expect the stack to be in trap order, with %rip being
```



```
581          * the last entry, so we'll need to restore all our regs.  On i86xpv
582          * we'll need to compensate for XPV_TRAP_POP.
583          *
584          * We're hard-coding the three cases where KMDB has installed permanent
585          * handlers, since after we KDI_RESTORE_REGS(), we don't have registers
586          * to work with; we can't use a global since other CPUs can easily pass
587          * through here at the same time.
588          *
589          * Note that we handle T_DBGENTR since userspace might have tried it.
590          */
591         movq    KRS_GREGS(%rdi), %rsp
592         movq    REG_OFF(KDIREG_TRAPNO)(%rsp), %rdi
593         cmpq    $T_SGLSTP, %rdi
594         je      1f
595         cmpq    $T_BPTFLT, %rdi
596         je      2f
597         cmpq    $T_DBGENTR, %rdi
598         je      3f
599         /*
600          * Hmm, unknown handler.  Somebody forgot to update this when they
601          * added a new trap interposition... try to drop back into kmdb.
602          */
603         int     $T_DBGENTR

605 #define CALL_TRAP_HANDLER(name) \
606         KDI_RESTORE_REGS(%rsp); \
607         /* Discard state, trapno, err */ \
608         addq    $REG_OFF(KDIREG_RIP), %rsp; \
609         XPV_TRAP_PUSH; \
610         jmp     %cs:name

612 1:
613         CALL_TRAP_HANDLER(dbgtrap)
614         /*NOTREACHED*/
615 2:
616         CALL_TRAP_HANDLER(brktrap)
617         /*NOTREACHED*/
618 3:
619         CALL_TRAP_HANDLER(invaltrap)
620         /*NOTREACHED*/

622         SET_SIZE(kdi_pass_to_kernel)
```

*_____unchanged_portion_omitted_*

```
692 #endif  /* !__lint */

694 #if defined(__lint)
695 /*ARGSUSED*/
696 void
697 kdi_cpu_debug_init(kdi_cpusave_t *save)
698 {
699 }
700 #else   /* __lint */

642         ENTRY_NP(kdi_cpu_debug_init)
643         pushq   %rbp
644         movq    %rsp, %rbp

646         pushq   %rbx            /* macro will clobber %rbx */
647         KDI_RESTORE_DEBUGGING_STATE
648         popq    %rbx

650         leave
651         ret
652         SET_SIZE(kdi_cpu_debug_init)
```

```
654 #define GETDREG(name, r)        \
655         ENTRY_NP(name);         \
656         movq    r, %rax;        \
657         ret;                    \
658         SET_SIZE(name)

660 #define SETDREG(name, r)        \
661         ENTRY_NP(name);         \
662         movq    %rdi, r;        \
663         ret;                    \
664         SET_SIZE(name)

666         GETDREG(kdi_getdr0, %dr0)
667         GETDREG(kdi_getdr1, %dr1)
668         GETDREG(kdi_getdr2, %dr2)
669         GETDREG(kdi_getdr3, %dr3)
670         GETDREG(kdi_getdr6, %dr6)
671         GETDREG(kdi_getdr7, %dr7)

673         SETDREG(kdi_setdr0, %dr0)
674         SETDREG(kdi_setdr1, %dr1)
675         SETDREG(kdi_setdr2, %dr2)
676         SETDREG(kdi_setdr3, %dr3)
677         SETDREG(kdi_setdr6, %dr6)
678         SETDREG(kdi_setdr7, %dr7)

713         SET_SIZE(kdi_cpu_debug_init)
680 #endif /* !__lint */
```

```
**********************************************************
   12124 Fri Apr  6 17:25:12 2018
new/usr/src/uts/intel/kdi/kdi_idt.c
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
9207 kdi_idt: Cast GATESEG_GETOFFSET through uintptr_t
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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

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

uint_t          kdi_msr_wrexit_msr;
uint64_t        *kdi_msr_wrexit_valp;

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);
```

```
117 extern idt_hdlr_f kdi_trap0, kdi_trap1, kdi_int2, kdi_trap3, kdi_trap4;
118 extern idt_hdlr_f kdi_trap5, kdi_trap6, kdi_trap7, kdi_trap9;
119 extern idt_hdlr_f kdi_traperr10, kdi_traperr11, kdi_traperr12;
120 extern idt_hdlr_f kdi_traperr13, kdi_traperr14, kdi_trap16, kdi_traperr17;
120 extern idt_hdlr_f kdi_traperr13, kdi_traperr14, kdi_trap16, kdi_trap17;
121 extern idt_hdlr_f kdi_trap18, kdi_trap19, kdi_trap20, kdi_ivct32;
122 extern idt_hdlr_f kdi_invaltrap;
123 extern size_t kdi_ivct_size;
124 extern char kdi_slave_entry_patch;

125 typedef struct kdi_gate_spec {
126         uint_t kgs_vec;
127         uint_t kgs_dpl;
128 } kdi_gate_spec_t;
______unchanged_portion_omitted_

139 static gate_desc_t kdi_kgates[KDI_GATE_NVECS];

141 extern gate_desc_t kdi_idt[NIDT];
142 gate_desc_t kdi_idt[NIDT];

143 struct idt_description {
144         uint_t id_low;
145         uint_t id_high;
146         idt_hdlr_f *id_basehdlr;
147         size_t *id_incrp;
148 } idt_description[] = {
149         { T_ZERODIV, 0,          kdi_trap0, NULL },
150         { T_SGLSTP, 0,           kdi_trap1, NULL },
151         { T_NMIFLT, 0,           kdi_int2, NULL },
152         { T_BPTFLT, 0,           kdi_trap3, NULL },
153         { T_OVFLW, 0,            kdi_trap4, NULL },
154         { T_BOUNDFLT, 0,         kdi_trap5, NULL },
155         { T_ILLINST, 0,          kdi_trap6, NULL },
156         { T_NOEXTFLT, 0,         kdi_trap7, NULL },
157 #if !defined(__xpv)
158         { T_DBLFLT, 0,           syserrtrap, NULL },
159 #endif
160         { T_EXTOVRFLT, 0,        kdi_trap9, NULL },
161         { T_TSSFLT, 0,           kdi_traperr10, NULL },
162         { T_SEGFLT, 0,           kdi_traperr11, NULL },
163         { T_STKFLT, 0,           kdi_traperr12, NULL },
164         { T_GPFLT, 0,            kdi_traperr13, NULL },
165         { T_PGFLT, 0,            kdi_traperr14, NULL },
166         { 15, 0,                 kdi_invaltrap, NULL },
167         { T_EXTERRFLT, 0,        kdi_trap16, NULL },
168         { T_ALIGNMENT, 0,        kdi_traperr17, NULL },
169         { T_ALIGNMENT, 0,        kdi_trap17, NULL },
169         { T_MCE, 0,              kdi_trap18, NULL },
170         { T_SIMDFPE, 0,          kdi_trap19, NULL },
171         { T_DBGENTR, 0,          kdi_trap20, NULL },
172         { 21, 31,                kdi_invaltrap, NULL },
173         { 32, 255,               kdi_ivct32, &kdi_ivct_size },
174         { 0, 0, NULL },
175 };

177 void
178 kdi_idt_init(selector_t sel)
179 {
180         struct idt_description *id;
181         int i;

183         for (id = idt_description; id->id_basehdlr != NULL; id++) {
184                 uint_t high = id->id_high != 0 ? id->id_high : id->id_low;
185                 size_t incr = id->id_incrp != NULL ? *id->id_incrp : 0;
```



```
187 #if !defined(__xpv)
188                 if (kpti_enable && sel == KCS_SEL && id->id_low == T_DBLFLT)
189                         id->id_basehdlr = tr_syserrtrap;
190 #endif

192                 for (i = id->id_low; i <= high; i++) {
193                         caddr_t hdlr = (caddr_t)id->id_basehdlr +
194                             incr * (i - id->id_low);
195                         set_gatesegd(&kdi_idt[i], (void (*)())hdlr, sel,
196                             SDT_SYSIGT, TRP_KPL, IST_DBG);
192                             SDT_SYSIGT, TRP_KPL, i);
197                 }
198         }
199 }

197 /*
198  * Patch caller-provided code into the debugger's IDT handlers.  This code is
199  * used to save MSRs that must be saved before the first branch.  All handlers
200  * are essentially the same, and end with a branch to kdi_cmnint.  To save the
201  * MSR, we need to patch in before the branch.  The handlers have the following
202  * structure: KDI_MSR_PATCHOFF bytes of code, KDI_MSR_PATCHSZ bytes of
203  * patchable space, followed by more code.
204  */
205 void
206 kdi_idt_patch(caddr_t code, size_t sz)
207 {
208         int i;

210         ASSERT(sz <= KDI_MSR_PATCHSZ);

212         for (i = 0; i < sizeof (kdi_idt) / sizeof (struct gate_desc); i++) {
213                 gate_desc_t *gd;
214                 uchar_t *patch;

216                 if (i == T_DBLFLT)
217                         continue;       /* uses kernel's handler */

219                 gd = &kdi_idt[i];
220                 patch = (uchar_t *)GATESEG_GETOFFSET(gd) + KDI_MSR_PATCHOFF;

222                 /*
223                  * We can't ASSERT that there's a nop here, because this may be
224                  * a debugger restart.  In that case, we're copying the new
225                  * patch point over the old one.
226                  */
227                 /* FIXME: dtrace fbt ... */
228                 bcopy(code, patch, sz);

230                 /* Fill the rest with nops to be sure */
231                 while (sz < KDI_MSR_PATCHSZ)
232                         patch[sz++] = 0x90; /* nop */
233         }
234 }

201 static void
202 kdi_idt_gates_install(selector_t sel, int saveold)
203 {
204         gate_desc_t gates[KDI_GATE_NVECS];
205         int i;

207         bzero(gates, sizeof (*gates));

209         for (i = 0; i < KDI_GATE_NVECS; i++) {
210                 const kdi_gate_spec_t *gs = &kdi_gate_specs[i];
211                 uintptr_t func = GATESEG_GETOFFSET(&kdi_idt[gs->kgs_vec]);
```

```
212                 set_gatesegd(&gates[i], (void (*)())func, sel, SDT_SYSIGT,
213                     gs->kgs_dpl, IST_DBG);
248                     gs->kgs_dpl, gs->kgs_vec);
214         }

216         for (i = 0; i < KDI_GATE_NVECS; i++) {
217                 uint_t vec = kdi_gate_specs[i].kgs_vec;

219                 if (saveold)
220                         kdi_kgates[i] = CPU->cpu_m.mcpu_idt[vec];

222                 kdi_idt_write(&gates[i], vec);
223         }
224 }
```

_unchanged_portion_omitted_

```
281 /*
282  * On some processors, we'll need to clear a certain MSR before proceeding into
283  * the debugger.  Complicating matters, this MSR must be cleared before we take
284  * any branches.  We have patch points in every trap handler, which will cover
285  * all entry paths for master CPUs.  We also have a patch point in the slave
286  * entry code.
287  */
288 static void
289 kdi_msr_add_clrentry(uint_t msr)
290 {
291 #ifdef __amd64
292         uchar_t code[] = {
293                 0x51, 0x50, 0x52,               /* pushq %rcx, %rax, %rdx */
294                 0xb9, 0x00, 0x00, 0x00, 0x00,   /* movl $MSRNUM, %ecx */
295                 0x31, 0xc0,                     /* clr %eax */
296                 0x31, 0xd2,                     /* clr %edx */
297                 0x0f, 0x30,                     /* wrmsr */
298                 0x5a, 0x58, 0x59                /* popq %rdx, %rax, %rcx */
299         };
300         uchar_t *patch = &code[4];
301 #else
302         uchar_t code[] = {
303                 0x60,                           /* pushal */
304                 0xb9, 0x00, 0x00, 0x00, 0x00,   /* movl $MSRNUM, %ecx */
305                 0x31, 0xc0,                     /* clr %eax */
306                 0x31, 0xd2,                     /* clr %edx */
307                 0x0f, 0x30,                     /* wrmsr */
308                 0x61                            /* popal */
309         };
310         uchar_t *patch = &code[2];
311 #endif

313         bcopy(&msr, patch, sizeof (uint32_t));

315         kdi_idt_patch((caddr_t)code, sizeof (code));

317         bcopy(code, &kdi_slave_entry_patch, sizeof (code));
318 }

320 static void
321 kdi_msr_add_wrexit(uint_t msr, uint64_t *valp)
322 {
323         kdi_msr_wrexit_msr = msr;
324         kdi_msr_wrexit_valp = valp;
325 }

246 void
328 kdi_set_debug_msrs(kdi_msr_t *msrs)
329 {
330         int nmsrs, i;
```



```
332         ASSERT(kdi_cpusave[0].krs_msr == NULL);

334         /* Look in CPU0's MSRs for any special MSRs. */
335         for (nmsrs = 0; msrs[nmsrs].msr_num != 0; nmsrs++) {
336                 switch (msrs[nmsrs].msr_type) {
337                 case KDI_MSR_CLEARENTRY:
338                         kdi_msr_add_clrentry(msrs[nmsrs].msr_num);
339                         break;

341                 case KDI_MSR_WRITEDELAY:
342                         kdi_msr_add_wrexit(msrs[nmsrs].msr_num,
343                             msrs[nmsrs].kdi_msr_valp);
344                         break;
345                 }
346         }

348         nmsrs++;

350         for (i = 0; i < kdi_ncpusave; i++)
351                 kdi_cpusave[i].krs_msr = &msrs[nmsrs * i];
352 }

354 void
247 kdi_update_drreg(kdi_drreg_t *drreg)
248 {
249         kdi_drreg = *drreg;
250 }
```

_unchanged_portion_omitted_

```
273 /*
274  * Activation for CPUs other than the boot CPU, called from that CPU's
275  * mp_startup().  We saved the kernel's descriptors when we initialized the
276  * boot CPU, so we don't want to do it again.  Saving the handlers from this
277  * CPU's IDT would actually be dangerous with the CPU initialization method in
278  * use at the time of this writing.  With that method, the startup code creates
279  * the IDTs for slave CPUs by copying the one used by the boot CPU, which has
280  * already been interposed upon by KMDB.  Were we to interpose again, we'd
281  * replace the kernel's descriptors with our own in the save area.  By not
282  * saving, but still overwriting, we'll work in the current world, and in any
283  * future world where the IDT is generated from scratch.
284  */
285 void
286 kdi_cpu_init(void)
287 {
288         kdi_idt_gates_install(KCS_SEL, KDI_IDT_NOSAVE);
289         /* Load the debug registers. */
397         /* Load the debug registers and MSRs */
290         kdi_cpu_debug_init(&kdi_cpusave[CPU->cpu_id]);
291 }
```

_unchanged_portion_omitted_

```
304 void
305 kdi_activate(kdi_main_t main, kdi_cpusave_t *cpusave, uint_t ncpusave)
306 {
307         int i;
308         cpuset_t cpuset;

310         CPUSET_ALL(cpuset);

312         kdi_cpusave = cpusave;
313         kdi_ncpusave = ncpusave;

315         kdi_kmdb_main = main;

317         for (i = 0; i < kdi_ncpusave; i++) {
```

```
318                 kdi_cpusave[i].krs_cpu_id = i;

320                 kdi_cpusave[i].krs_curcrumb =
321                     &kdi_cpusave[i].krs_crumbs[KDI_NCRUMBS - 1];
322                 kdi_cpusave[i].krs_curcrumbidx = KDI_NCRUMBS - 1;
323         }

325         if (boothowto & RB_KMDB)
326                 kdi_idt_init(KMDBCODE_SEL);
327         else
328                 kdi_idt_init(KCS_SEL);

330         /* The initial selector set.  Updated by the debugger-entry code */
331 #ifndef __amd64
332         kdi_cs = B32CODE_SEL;
333         kdi_ds = kdi_fs = kdi_gs = B32DATA_SEL;
334 #endif

336         kdi_memranges[0].mr_base = kdi_segdebugbase;
337         kdi_memranges[0].mr_lim = kdi_segdebugbase + kdi_segdebugsize - 1;
338         kdi_nmemranges = 1;

340         kdi_drreg.dr_ctl = KDIREG_DRCTL_RESERVED;
341         kdi_drreg.dr_stat = KDIREG_DRSTAT_RESERVED;

451         kdi_msr_wrexit_msr = 0;
452         kdi_msr_wrexit_valp = NULL;

343         if (boothowto & RB_KMDB) {
344                 kdi_idt_gates_install(KMDBCODE_SEL, KDI_IDT_NOSAVE);
345         } else {
346                 xc_call(0, 0, 0, CPUSET2BV(cpuset),
347                     (xc_func_t)kdi_cpu_activate);
348         }
349 }
```

_____unchanged_portion_omitted_

```
368 /*
369  * We receive all breakpoints and single step traps.  Some of them,
370  * including those from userland and those induced by DTrace providers,
371  * are intended for the kernel, and must be processed there.  We adopt
372  * this ours-until-proven-otherwise position due to the painful
373  * consequences of sending the kernel an unexpected breakpoint or
374  * single step.  Unless someone can prove to us that the kernel is
375  * prepared to handle the trap, we'll assume there's a problem and will
376  * give the user a chance to debug it.
377  */
378 int
379 kdi_trap_pass(kdi_cpusave_t *cpusave)
380 {
381         greg_t tt = cpusave->krs_gregs[KDIREG_TRAPNO];
382         greg_t pc = cpusave->krs_gregs[KDIREG_PC];
383         greg_t cs = cpusave->krs_gregs[KDIREG_CS];

385         if (USERMODE(cs))
386                 return (1);

388         if (tt != T_BPTFLT && tt != T_SGLSTP)
389                 return (0);

391         if (tt == T_BPTFLT && kdi_dtrace_get_state() ==
392             KDI_DTSTATE_DTRACE_ACTIVE)
393                 return (1);

395         /*
396          * See the comments in the kernel's T_SGLSTP handler for why we need to
```



```
397          * do this.
398          */
399 #if !defined(__xpv)
400         if (tt == T_SGLSTP &&
401             (pc == (greg_t)sys_sysenter || pc == (greg_t)brand_sys_sysenter ||
402             pc == (greg_t)tr_sys_sysenter ||
403             pc == (greg_t)tr_brand_sys_sysenter)) {
404 #else
405         if (tt == T_SGLSTP &&
406             (pc == (greg_t)sys_sysenter || pc == (greg_t)brand_sys_sysenter)) {
407 #endif
511             (pc == (greg_t)sys_sysenter || pc == (greg_t)brand_sys_sysenter))
408                 return (1);
409         }

411         return (0);
412 }
```

_____unchanged_portion_omitted_

```
**********************************************************
    8885 Fri Apr  6 17:25:13 2018
new/usr/src/uts/intel/kdi/kdi_idthdl.s
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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"

/*
 * Companion to kdi_asm.s - the implementation of the trap and interrupt
 * Companion to kdi_idt.c - the implementation of the trap and interrupt
 * handlers.  For the most part, these handlers do the same thing - they
 * push a trap number onto the stack, followed by a jump to kdi_cmnint.
 * Each trap and interrupt has its own handler because each one pushes a
 * different number.
 */

#if defined(__lint)
#include <sys/types.h>
#else

#include <sys/asm_linkage.h>
#include <sys/asm_misc.h>
#include <sys/machprivregs.h>
#include <sys/privregs.h>
#include <sys/kdi_regs.h>
#include <sys/trap.h>
#include <sys/param.h>

#include <kdi_assym.h>
#include <assym.h>
/* Nothing in this file is of interest to lint. */
#if !defined(__lint)
```



```
/*
 * The default ASM_ENTRY_ALIGN (16) wastes far too much space.
 * The default ASM_ENTRY_ALIGN (16) wastes far too much space.  Pay no
 * attention to the fleet of nop's we're adding to each handler.
 */
#undef  ASM_ENTRY_ALIGN
#define ASM_ENTRY_ALIGN 8

/*
 * Generic trap and interrupt handlers.
 * We need the .align in ENTRY_NP (defined to be ASM_ENTRY_ALIGN) to match our
 * manual .align (KDI_MSR_PATCHOFF) in order to ensure that the space reserved
 * at the beginning of the handler for code is exactly KDI_MSR_PATCHOFF bytes
 * long.  Note that the #error below isn't supported by the preprocessor invoked
 * by as(1), and won't stop the build, but it'll emit a noticeable error message
 * which won't escape the filters.
 */
#if ASM_ENTRY_ALIGN != KDI_MSR_PATCHOFF
#error "ASM_ENTRY_ALIGN != KDI_MSR_PATCHOFF"
this won't assemble
#endif

#if defined(__xpv)
/*
 * kdi_idt_patch will, on certain processors, replace the patch points below
 * with MSR-clearing code.  kdi_id_patch has intimate knowledge of the size of
 * the nop hole, as well as the structure of the handlers.  Do not change
 * anything here without also changing kdi_idt_patch.
 */

#define INTERRUPT_TRAMPOLINE
/*
 * Generic trap and interrupt handlers.
 */

#else
#if defined(__xpv) && defined(__amd64)

/*
 * If we're !xpv, then we will need to support KPTI (kernel page table
 * isolation), where we have separate page tables for user and kernel modes.
 * There's more detail about this in kpti_trampolines.s and hat_i86.c
 * The hypervisor places r11 and rcx on the stack.
 */

#define INTERRUPT_TRAMPOLINE                    \
        pushq   %r13;                           \
        pushq   %r14;                           \
        subq    $KPTI_R14, %rsp;                \
        /* Check for clobbering */              \
        cmp     $0, KPTI_FLAG(%rsp);            \
        je      1f;                             \
        /* Don't worry, this totally works */   \
        int     $8;                             \
1:                                              \
        movq    $1, KPTI_FLAG(%rsp);            \
        /* Save current %cr3. */                \
        mov     %cr3, %r14;                     \
        mov     %r14, KPTI_TR_CR3(%rsp);        \
        /* Switch to paranoid %cr3. */          \
        mov     kpti_safe_cr3, %r14;            \
        mov     %r14, %cr3;                     \
                                                \
        cmpw    $KCS_SEL, KPTI_CS(%rsp);        \
        je      3f;                             \
```

```
 93 2:                                          \
 94         /* Get our cpu_t in %r13 */         \
 95         mov     %rsp, %r13;                 \
 96         and     $(~(MMU_PAGESIZE - 1)), %r13;   \
 97         subq    $CPU_KPTI_START, %r13;      \
 98         /* Use top of the kthread stk */    \
 99         mov     CPU_THREAD(%r13), %r14;     \
100         mov     T_STACK(%r14), %r14;        \
101         addq    $REGSIZE+MINFRAME, %r14;    \
102         jmp     5f;                         \
103 3:                                          \
104         /* Check the %rsp in the frame. */  \
105         /* Is it above kernel base? */      \
106         mov     kpti_kbase, %r14;           \
107         cmp     %r14, KPTI_RSP(%rsp);       \
108         jb      2b;                         \
109         /* Is it within the kpti_frame page? */ \
110         mov     %rsp, %r13;                 \
111         and     $(~(MMU_PAGESIZE - 1)), %r13;   \
112         mov     KPTI_RSP(%rsp), %r14;       \
113         and     $(~(MMU_PAGESIZE - 1)), %r14;   \
114         cmp     %r13, %r14;                 \
115         je      2b;                         \
116         /* Use the %rsp from the trap frame. */ \
117         /* We already did %cr3. */          \
118         mov     KPTI_RSP(%rsp), %r14;       \
119         and     $(~0xf), %r14;              \
120 5:                                          \
121         mov     %rsp, %r13;                 \
122         /* %r14 contains our destination stk */ \
123         mov     %r14, %rsp;                 \
124         pushq   KPTI_SS(%r13);              \
125         pushq   KPTI_RSP(%r13);             \
126         pushq   KPTI_RFLAGS(%r13);          \
127         pushq   KPTI_CS(%r13);              \
128         pushq   KPTI_RIP(%r13);             \
129         pushq   KPTI_ERR(%r13);             \
130         mov     KPTI_R14(%r13), %r14;       \
131         movq    $0, KPTI_FLAG(%r13);        \
132         mov     KPTI_R13(%r13), %r13
 79 #define TRAP_NOERR(trapno) \
 80         popq    %rcx;           \
 81         popq    %r11;           \
 82         pushq   $trapno

134 #endif  /* !__xpv */
 84 #define TRAP_ERR(trapno)        \
 85         popq    %rcx;           \
 86         popq    %r11;           \
 87         pushq   $0;             \
 88         pushq   $trapno

 90 #else

 92 #define TRAP_NOERR(trapno)      \
 93         push    $trapno

 95 #define TRAP_ERR(trapno)        \
 96         push    $0;             \
 97         push    $trapno

 99 #endif  /* __xpv && __amd64 */


137 #define MKIVCT(n) \
138         ENTRY_NP(kdi_ivct/**/n/**/);    \
```



```
139         XPV_TRAP_POP;                   \
140         push    $0; /* err */          \
141         INTERRUPT_TRAMPOLINE;           \
142         push    $n;                     \
104         TRAP_ERR(n);                    \
105         .align  KDI_MSR_PATCHOFF;       \
106         KDI_MSR_PATCH;                  \
143         jmp     kdi_cmnint;             \
144         SET_SIZE(kdi_ivct/**/n/**/)

146 #define MKTRAPHDLR(n) \
147         ENTRY_NP(kdi_trap/**/n);        \
148         XPV_TRAP_POP;                   \
149         push    $0; /* err */          \
150         INTERRUPT_TRAMPOLINE;           \
151         push    $n;                     \
112         TRAP_ERR(n);                    \
113         .align  KDI_MSR_PATCHOFF;       \
114         KDI_MSR_PATCH;                  \
152         jmp     kdi_cmnint;             \
153         SET_SIZE(kdi_trap/**/n/**/)

155 #define MKTRAPERRHDLR(n) \
156         ENTRY_NP(kdi_traperr/**/n);     \
157         XPV_TRAP_POP;                   \
158         INTERRUPT_TRAMPOLINE;           \
159         push    $n;                     \
120         TRAP_NOERR(n);                  \
121         .align  KDI_MSR_PATCHOFF;       \
122         KDI_MSR_PATCH;                  \
160         jmp     kdi_cmnint;             \
161         SET_SIZE(kdi_traperr/**/n)

163 #if !defined(__xpv)
164 #define MKNMIHDLR \
165         ENTRY_NP(kdi_int2);             \
166         push    $0;                     \
167         push    $2;                     \
168         pushq   %r13;                   \
169         mov     kpti_safe_cr3, %r13;    \
170         mov     %r13, %cr3;             \
171         popq    %r13;                   \
128         TRAP_NOERR(2);                  \
129         .align  KDI_MSR_PATCHOFF;       \
130         KDI_MSR_PATCH;                  \
172         jmp     kdi_nmiint;             \
173         SET_SIZE(kdi_int2)

175 #define MKMCEHDLR \
176         ENTRY_NP(kdi_trap18);           \
177         push    $0;                     \
178         push    $18;                    \
179         pushq   %r13;                   \
180         mov     kpti_safe_cr3, %r13;    \
181         mov     %r13, %cr3;             \
182         popq    %r13;                   \
183         jmp     kdi_cmnint;             \
184         SET_SIZE(kdi_trap18)
185 #else
186 #define MKNMIHDLR \
187         ENTRY_NP(kdi_int2);             \
188         push    $0;                     \
189         push    $2;                     \
190         jmp     kdi_nmiint;             \
191         SET_SIZE(kdi_int2)
```

```
193 #define MKMCEHDLR \
194         ENTRY_NP(kdi_trap18);           \
195         push    $0;                     \
196         push    $18;                    \
197         jmp     kdi_cmnint;             \
198         SET_SIZE(kdi_trap18)
199 #endif

201 /*
202  * The only way we should reach here is by an explicit "int 0x.." which is
203  * defined not to push an error code.
204  */
205 #define MKINVALHDLR \
206         ENTRY_NP(kdi_invaltrap);        \
207         XPV_TRAP_POP;                   \
208         push    $0; /* err */          \
209         INTERRUPT_TRAMPOLINE;           \
210         push    $255;                   \
136         TRAP_NOERR(255);                \
137         .align  KDI_MSR_PATCHOFF;       \
138         KDI_MSR_PATCH;                  \
211         jmp     kdi_cmnint;             \
212         SET_SIZE(kdi_invaltrap)

214         .data
215         DGDEF3(kdi_idt, 16 * NIDT, MMU_PAGESIZE)
216         .fill   MMU_PAGESIZE, 1, 0

218 #if !defined(__xpv)
219 .section ".text"
220 .align MMU_PAGESIZE
221 .global kdi_isr_start
222 kdi_isr_start:
223         nop

225 .global kpti_safe_cr3
226 .global kpti_kbase
227 #endif

229 /*
230  * The handlers themselves
231  */

233         MKINVALHDLR
234         MKTRAPHDLR(0)
235         MKTRAPHDLR(1)
236         MKNMIHDLR/*2*/
237         MKTRAPHDLR(3)
238         MKTRAPHDLR(4)
239         MKTRAPHDLR(5)
240         MKTRAPHDLR(6)
241         MKTRAPHDLR(7)
242         MKTRAPHDLR(9)
243         MKTRAPHDLR(15)
244         MKTRAPHDLR(16)
245         MKMCEHDLR/*18*/
158         MKTRAPHDLR(17)
159         MKTRAPHDLR(18)
246         MKTRAPHDLR(19)
247         MKTRAPHDLR(20)

249         MKTRAPERRHDLR(8)
250         MKTRAPERRHDLR(10)
251         MKTRAPERRHDLR(11)
252         MKTRAPERRHDLR(12)
253         MKTRAPERRHDLR(13)
```



```
254         MKTRAPERRHDLR(14)
255         MKTRAPERRHDLR(17)

257         .globl  kdi_ivct_size
258 kdi_ivct_size:
259         .NWORD [kdi_ivct33-kdi_ivct32]

261         /* 10 billion and one interrupt handlers */
262 kdi_ivct_base:
263         MKIVCT(32);     MKIVCT(33);     MKIVCT(34);     MKIVCT(35);
264         MKIVCT(36);     MKIVCT(37);     MKIVCT(38);     MKIVCT(39);
265         MKIVCT(40);     MKIVCT(41);     MKIVCT(42);     MKIVCT(43);
266         MKIVCT(44);     MKIVCT(45);     MKIVCT(46);     MKIVCT(47);
267         MKIVCT(48);     MKIVCT(49);     MKIVCT(50);     MKIVCT(51);
268         MKIVCT(52);     MKIVCT(53);     MKIVCT(54);     MKIVCT(55);
269         MKIVCT(56);     MKIVCT(57);     MKIVCT(58);     MKIVCT(59);
270         MKIVCT(60);     MKIVCT(61);     MKIVCT(62);     MKIVCT(63);
271         MKIVCT(64);     MKIVCT(65);     MKIVCT(66);     MKIVCT(67);
272         MKIVCT(68);     MKIVCT(69);     MKIVCT(70);     MKIVCT(71);
273         MKIVCT(72);     MKIVCT(73);     MKIVCT(74);     MKIVCT(75);
274         MKIVCT(76);     MKIVCT(77);     MKIVCT(78);     MKIVCT(79);
275         MKIVCT(80);     MKIVCT(81);     MKIVCT(82);     MKIVCT(83);
276         MKIVCT(84);     MKIVCT(85);     MKIVCT(86);     MKIVCT(87);
277         MKIVCT(88);     MKIVCT(89);     MKIVCT(90);     MKIVCT(91);
278         MKIVCT(92);     MKIVCT(93);     MKIVCT(94);     MKIVCT(95);
279         MKIVCT(96);     MKIVCT(97);     MKIVCT(98);     MKIVCT(99);
280         MKIVCT(100);    MKIVCT(101);    MKIVCT(102);    MKIVCT(103);
281         MKIVCT(104);    MKIVCT(105);    MKIVCT(106);    MKIVCT(107);
282         MKIVCT(108);    MKIVCT(109);    MKIVCT(110);    MKIVCT(111);
283         MKIVCT(112);    MKIVCT(113);    MKIVCT(114);    MKIVCT(115);
284         MKIVCT(116);    MKIVCT(117);    MKIVCT(118);    MKIVCT(119);
285         MKIVCT(120);    MKIVCT(121);    MKIVCT(122);    MKIVCT(123);
286         MKIVCT(124);    MKIVCT(125);    MKIVCT(126);    MKIVCT(127);
287         MKIVCT(128);    MKIVCT(129);    MKIVCT(130);    MKIVCT(131);
288         MKIVCT(132);    MKIVCT(133);    MKIVCT(134);    MKIVCT(135);
289         MKIVCT(136);    MKIVCT(137);    MKIVCT(138);    MKIVCT(139);
290         MKIVCT(140);    MKIVCT(141);    MKIVCT(142);    MKIVCT(143);
291         MKIVCT(144);    MKIVCT(145);    MKIVCT(146);    MKIVCT(147);
292         MKIVCT(148);    MKIVCT(149);    MKIVCT(150);    MKIVCT(151);
293         MKIVCT(152);    MKIVCT(153);    MKIVCT(154);    MKIVCT(155);
294         MKIVCT(156);    MKIVCT(157);    MKIVCT(158);    MKIVCT(159);
295         MKIVCT(160);    MKIVCT(161);    MKIVCT(162);    MKIVCT(163);
296         MKIVCT(164);    MKIVCT(165);    MKIVCT(166);    MKIVCT(167);
297         MKIVCT(168);    MKIVCT(169);    MKIVCT(170);    MKIVCT(171);
298         MKIVCT(172);    MKIVCT(173);    MKIVCT(174);    MKIVCT(175);
299         MKIVCT(176);    MKIVCT(177);    MKIVCT(178);    MKIVCT(179);
300         MKIVCT(180);    MKIVCT(181);    MKIVCT(182);    MKIVCT(183);
301         MKIVCT(184);    MKIVCT(185);    MKIVCT(186);    MKIVCT(187);
302         MKIVCT(188);    MKIVCT(189);    MKIVCT(190);    MKIVCT(191);
303         MKIVCT(192);    MKIVCT(193);    MKIVCT(194);    MKIVCT(195);
304         MKIVCT(196);    MKIVCT(197);    MKIVCT(198);    MKIVCT(199);
305         MKIVCT(200);    MKIVCT(201);    MKIVCT(202);    MKIVCT(203);
306         MKIVCT(204);    MKIVCT(205);    MKIVCT(206);    MKIVCT(207);
307         MKIVCT(208);    MKIVCT(209);    MKIVCT(210);    MKIVCT(211);
308         MKIVCT(212);    MKIVCT(213);    MKIVCT(214);    MKIVCT(215);
309         MKIVCT(216);    MKIVCT(217);    MKIVCT(218);    MKIVCT(219);
310         MKIVCT(220);    MKIVCT(221);    MKIVCT(222);    MKIVCT(223);
311         MKIVCT(224);    MKIVCT(225);    MKIVCT(226);    MKIVCT(227);
312         MKIVCT(228);    MKIVCT(229);    MKIVCT(230);    MKIVCT(231);
313         MKIVCT(232);    MKIVCT(233);    MKIVCT(234);    MKIVCT(235);
314         MKIVCT(236);    MKIVCT(237);    MKIVCT(238);    MKIVCT(239);
315         MKIVCT(240);    MKIVCT(241);    MKIVCT(242);    MKIVCT(243);
316         MKIVCT(244);    MKIVCT(245);    MKIVCT(246);    MKIVCT(247);
317         MKIVCT(248);    MKIVCT(249);    MKIVCT(250);    MKIVCT(251);
318         MKIVCT(252);    MKIVCT(253);    MKIVCT(254);    MKIVCT(255);
```

```
320 #if !defined(__xpv)
321 .section ".text"
322 .align MMU_PAGESIZE
323 .global kdi_isr_end
324 kdi_isr_end:
325         nop
326 #endif

328 #endif /* !__lint */
```

```
**********************************************************
    1740 Fri Apr  6 17:25:13 2018
new/usr/src/uts/intel/kdi/kdi_offsets.in
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
\
\ Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
\ Use is subject to license terms.
\
\ Copyright 2018 Joyent, Inc.
\
\ 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
\
\ ident "%Z%%M% %I%     %E% SMI"
\
\ CPU-save structure offsets for use in assembly code.
\
\ Keep in sync with kdi_state.h
\

#include <sys/cpuvar.h>
#include <sys/kdi_impl.h>

kdi_memrange_t  MR_SIZE
                mr_base
                mr_lim

kdi_crumb_t     KRM_SIZE
                krm_cpu_state
                krm_pc
                krm_sp
                krm_trapno
                krm_flag

kdi_drreg_t
                dr_ctl
                dr_stat
                dr_addr

kdi_msr_t       MSR_SIZE
                msr_num
                msr_type
```



```
                _u._msr_valp            MSR_VALP
                _u._msr_val             MSR_VAL

kdi_cpusave_t   KRS_SIZE
                krs_gregs
                krs_dr
                krs_dr.dr_ctl           KRS_DRCTL
                krs_dr.dr_stat          KRS_DRSTAT
                krs_gdt
                krs_idt
                krs_cr0
                krs_msr
                krs_cpu_state
                krs_curcrumbidx
                krs_curcrumb
                krs_crumbs

cpu
                cpu_id

greg_t          KREG_SIZE

#if defined(__amd64)
\#define        REG_SHIFT               3
#else
\#define        REG_SHIFT               2
#endif

\#define        DRADDR_IDX(num)         _CONST(_MUL(num, DR_ADDR_INCR))
\#define        DRADDR_OFF(num)         _CONST(DRADDR_IDX(num) + DR_ADDR)
\#define        KRS_DROFF(num)          _CONST(DRADDR_OFF(num) + KRS_DR)
\#define        REG_OFF(reg)            _CONST(_CONST(reg) << REG_SHIFT)
\#define        KDIREG_OFF(reg)         _CONST(_MUL(KREG_SIZE, reg) + KRS_GREGS)
```

```
**********************************************************
    4230 Fri Apr  6 17:25:13 2018
new/usr/src/uts/intel/os/arch_kdi.c
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
/*
 * 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.
 */

/*
 * Kernel/Debugger Interface (KDI) routines.  Called during debugger under
 * various system states (boot, while running, while the debugger has control).
 * Functions intended for use while the debugger has control may not grab any
 * locks or perform any functions that assume the availability of other system
 * services.
 */

#include <sys/systm.h>
#include <sys/x86_archext.h>
#include <sys/kdi_impl.h>
#include <sys/smp_impldefs.h>
#include <sys/psm_types.h>
#include <sys/segments.h>
#include <sys/archsystm.h>
#include <sys/controlregs.h>
#include <sys/trap.h>
#include <sys/kobj.h>
#include <sys/kobj_impl.h>
#include <sys/clock_impl.h>

static void
kdi_system_claim(void)
{
        lbolt_debug_entry();

        psm_notifyf(PSM_DEBUG_ENTER);
}
_____unchanged_portion_omitted_
```



```
/*
 * On Intel, most of these are shared between i86*, so this is really an
 * arch_kdi_init().
 */
void
mach_kdi_init(kdi_t *kdi)
{
        kdi->kdi_plat_call = kdi_plat_call;
        kdi->kdi_kmdb_enter = kmdb_enter;
        kdi->mkdi_activate = kdi_activate;
        kdi->mkdi_deactivate = kdi_deactivate;
        kdi->mkdi_idt_switch = kdi_idt_switch;
        kdi->mkdi_update_drreg = kdi_update_drreg;
        kdi->mkdi_set_debug_msrs = kdi_set_debug_msrs;
        kdi->mkdi_get_userlimit = kdi_get_userlimit;
        kdi->mkdi_get_cpuinfo = kdi_get_cpuinfo;
        kdi->mkdi_stop_slaves = kdi_stop_slaves;
        kdi->mkdi_start_slaves = kdi_start_slaves;
        kdi->mkdi_slave_wait = kdi_slave_wait;
        kdi->mkdi_memrange_add = kdi_memrange_add;
        kdi->mkdi_reboot = kdi_reboot;
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    6372 Fri Apr  6 17:25:13 2018
new/usr/src/uts/intel/sys/archsystm.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@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 (c) 1993, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2018 Joyent, Inc.
 * Copyright 2017 Joyent, Inc.
 */

#ifndef _SYS_ARCHSYSTM_H
#define _SYS_ARCHSYSTM_H

/*
 * A selection of ISA-dependent interfaces
 */

#include <vm/seg_enum.h>
#include <vm/page.h>

#ifdef __cplusplus
extern "C" {
#endif

#ifdef _KERNEL

extern greg_t getfp(void);
extern int getpil(void);

extern ulong_t getcr0(void);
extern void setcr0(ulong_t);
extern ulong_t getcr8(void);
extern void setcr8(ulong_t);
extern ulong_t getcr2(void);
extern void clflush_insn(caddr_t addr);
extern void mfence_insn(void);

#if defined(__i386)
extern uint16_t getgs(void);
extern void setgs(uint16_t);

extern void patch_sse(void);
```



```
extern void patch_sse2(void);
#endif

extern void patch_xsave(void);
extern kmem_cache_t *fpsave_cachep;

extern void cli(void);
extern void sti(void);

extern void tenmicrosec(void);

extern void restore_int_flag(ulong_t);
extern void intr_restore(ulong_t);
extern ulong_t clear_int_flag(void);
extern ulong_t intr_clear(void);
extern ulong_t getflags(void);
extern int interrupts_enabled(void);

extern void int3(void);
extern void int18(void);
extern void int20(void);
extern void int_cmci(void);

#if defined(__amd64)
extern void sys_syscall(), tr_sys_syscall();
extern void sys_syscall32(), tr_sys_syscall32();
extern void sys_syscall();
extern void sys_syscall32();
extern void sys_lcall32();
extern void sys_syscall_int();
extern void tr_sys_syscall_int();
extern void brand_sys_syscall(), tr_brand_sys_syscall();
extern void brand_sys_syscall32(), tr_brand_sys_syscall32();
extern void brand_sys_syscall_int(), tr_brand_sys_syscall_int();
extern void brand_sys_syscall();
extern void brand_sys_syscall32();
extern void brand_sys_syscall_int();
extern int update_sregs();
extern void reset_sregs();
#elif defined(__i386)
extern void sys_call();
extern void tr_sys_call();
extern void brand_sys_call();
#endif
extern void sys_sysenter();
extern void tr_sys_sysenter();
extern void _sys_sysenter_post_swapgs();
extern void brand_sys_sysenter();
extern void tr_brand_sys_sysenter();
extern void _brand_sys_sysenter_post_swapgs();

extern void dosyscall(void);

extern void bind_hwcap(void);

extern uint16_t inw(int port);
extern uint32_t inl(int port);
extern void outw(int port, uint16_t value);
extern void outl(int port, uint32_t value);

extern void pc_reset(void) __NORETURN;
extern void efi_reset(void) __NORETURN;
extern void reset(void) __NORETURN;
extern int goany(void);

extern void setgregs(klwp_t *, gregset_t);
```

```
120 extern void getgregs(klwp_t *, gregset_t);
121 extern void setfpregs(klwp_t *, fpregset_t *);
122 extern void getfpregs(klwp_t *, fpregset_t *);

124 #if defined(_SYSCALL32_IMPL)
125 extern void getgregs32(klwp_t *, gregset32_t);
126 extern void setfpregs32(klwp_t *, fpregset32_t *);
127 extern void getfpregs32(klwp_t *, fpregset32_t *);
128 #endif

130 struct fpu_ctx;

132 extern void fp_free(struct fpu_ctx *, int);
133 extern void fp_save(struct fpu_ctx *);
134 extern void fp_restore(struct fpu_ctx *);

136 extern int fpu_pentium_fdivbug;

138 extern void sep_save(void *);
139 extern void sep_restore(void *);

141 extern void brand_interpositioning_enable(void);
142 extern void brand_interpositioning_disable(void);

144 struct regs;

146 extern int instr_size(struct regs *, caddr_t *, enum seg_rw);

148 extern int enable_cbcp; /* patchable in /etc/system */

150 extern uint_t cpu_hwcap_flags;
151 extern uint_t cpu_freq;
152 extern uint64_t cpu_freq_hz;

154 extern int use_sse_pagecopy;
155 extern int use_sse_pagezero;
156 extern int use_sse_copy;

158 extern caddr_t i86devmap(pfn_t, pgcnt_t, uint_t);
159 extern page_t *page_numtopp_alloc(pfn_t pfnum);

161 extern void hwblkclr(void *, size_t);
162 extern void hwblkpagecopy(const void *, void *);
163 extern void page_copy_no_xmm(void *dst, void *src);
164 extern void block_zero_no_xmm(void *dst, int len);
165 #define BLOCKZEROALIGN (4 * sizeof (void *))

167 extern void (*kcpc_hw_enable_cpc_intr)(void);

169 extern void init_desctbls(void);

171 extern user_desc_t *cpu_get_gdt(void);

173 extern void switch_sp_and_call(void *, void (*)(uint_t, uint_t), uint_t,
174         uint_t);
175 extern hrtime_t (*gethrtimef)(void);
176 extern hrtime_t (*gethrtimeunscaledf)(void);
177 extern void (*scalehrtimef)(hrtime_t *);
178 extern uint64_t (*unscalehrtimef)(hrtime_t);
179 extern void (*gethrestimef)(timestruc_t *);

181 extern void av_dispatch_softvect(uint_t);
182 extern void av_dispatch_autovect(uint_t);
183 extern uint_t atomic_btr32(uint32_t *, uint_t);
184 extern uint_t bsrw_insn(uint16_t);
185 extern int sys_rtt_common(struct regs *);
```



```
186 extern void fakesoftint(void);

188 extern void *plat_traceback(void *);

190 /*
191  * The following two macros are the four byte instruction sequence of stac, ret
192  * and clac, ret. These are used in startup_smap() as a part of properly setting
193  * up the valid instructions. For more information on SMAP, see
194  * uts/intel/ia32/ml/copy.s.
195  */
196 #define SMAP_CLAC_INSTR 0xc3ca010f
197 #define SMAP_STAC_INSTR 0xc3cb010f
198 extern void smap_disable(void);
199 extern void smap_enable(void);

201 #if defined(__xpv)
202 extern void xen_init_callbacks(void);
203 extern void xen_set_callback(void (*)(void), uint_t, uint_t);
204 extern void xen_printf(const char *, ...);
205 #define cpr_dprintf xen_printf
206 extern int xpv_panicking;
207 #define IN_XPV_PANIC() (xpv_panicking > 0)
208 #else
209 extern void setup_mca(void);
210 extern void pat_sync(void);
211 extern void patch_tsc_read(int);
212 #if defined(__amd64) && !defined(__xpv)
213 extern void patch_memops(uint_t);
214 #endif  /* defined(__amd64) && !defined(__xpv) */
215 extern void setup_xfem(void);
216 #define cpr_dprintf prom_printf
217 #define IN_XPV_PANIC() (__lintzero)
218 #endif

220 #endif /* _KERNEL */

222 #if defined(_KERNEL) || defined(_BOOT)
223 extern uint8_t inb(int port);
224 extern void outb(int port, uint8_t value);
225 #endif

227 #ifdef __cplusplus
228 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    8003 Fri Apr  6 17:25:13 2018
new/usr/src/uts/intel/sys/controlregs.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
  1 /*
  2  * CDDL HEADER START
  3  *
  4  * The contents of this file are subject to the terms of the
  5  * Common Development and Distribution License (the "License").
  6  * You may not use this file except in compliance with the License.
  7  *
  8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  9  * or http://www.opensolaris.org/os/licensing.
 10  * See the License for the specific language governing permissions
 11  * and limitations under the License.
 12  *
 13  * When distributing Covered Code, include this CDDL HEADER in each
 14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 15  * If applicable, add the following below this CDDL HEADER, with the
 16  * fields enclosed by brackets "[]" replaced with your own identifying
 17  * information: Portions Copyright [yyyy] [name of copyright owner]
 18  *
 19  * CDDL HEADER END
 20  */
 21 /*
 22  * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 23  * Copyright 2018, Joyent, Inc.
 23  * Copyright 2015, Joyent, Inc.
 24  */

 26 #ifndef _SYS_CONTROLREGS_H
 27 #define _SYS_CONTROLREGS_H

 29 #ifndef _ASM
 30 #include <sys/types.h>
 31 #endif

 33 #ifdef __cplusplus
 34 extern "C" {
 35 #endif

 37 /*
 38  * This file describes the x86 architecture control registers which
 39  * are part of the privileged architecture.
 40  *
 41  * Many of these definitions are shared between IA-32-style and
 42  * AMD64-style processors.
 43  */

 45 /* CR0 Register */

 47 #define CR0_PG  0x80000000              /* paging enabled       */
 48 #define CR0_CD  0x40000000              /* cache disable        */
 49 #define CR0_NW  0x20000000              /* not writethrough     */
 50 #define CR0_AM  0x00040000              /* alignment mask       */
 51 #define CR0_WP  0x00010000              /* write protect        */
 52 #define CR0_NE  0x00000020              /* numeric error        */
 53 #define CR0_ET  0x00000010              /* extension type       */
```



```
 54 #define CR0_TS  0x00000008              /* task switch          */
 55 #define CR0_EM  0x00000004              /* emulation            */
 56 #define CR0_MP  0x00000002              /* monitor coprocessor  */
 57 #define CR0_PE  0x00000001              /* protection enabled   */

 59 /* XX64 eliminate these compatibility defines */

 61 #define CR0_CE  CR0_CD
 62 #define CR0_WT  CR0_NW

 64 #define FMT_CR0 \
 65         "\20\40pg\37cd\36nw\35am\21wp\6ne\5et\4ts\3em\2mp\1pe"

 67 /*
 68  * Set the FPU-related control bits to explain to the processor that
 69  * we're managing FPU state:
 70  * - set monitor coprocessor (allow TS bit to control FPU)
 71  * - set numeric exception (disable IGNNE# mechanism)
 72  * - set task switch (#nm on first fp instruction)
 73  * - clear emulate math bit (cause we're not emulating!)
 74  */
 75 #define CR0_ENABLE_FPU_FLAGS(cr)        \
 76         (((cr) | CR0_MP | CR0_NE | CR0_TS) & (uint32_t)~CR0_EM)

 78 /*
 79  * Set the FPU-related control bits to explain to the processor that
 80  * we're -not- managing FPU state:
 81  * - set emulate (all fp instructions cause #nm)
 82  * - clear monitor coprocessor (so fwait/wait doesn't #nm)
 83  */
 84 #define CR0_DISABLE_FPU_FLAGS(cr)       \
 85         (((cr) | CR0_EM) & (uint32_t)~CR0_MP)

 87 /* CR3 Register */

 89 #define CR3_PCD 0x00000010              /* cache disable                */
 90 #define CR3_PWT 0x00000008              /* write through                */
 91 #if defined(_ASM)
 92 #define CR3_NOINVL_BIT  0x8000000000000000
 93 #else
 94 #define CR3_NOINVL_BIT  0x8000000000000000ULL /* no invalidation        */
 95 #endif
 96 #define PCID_NONE       0x000           /* generic PCID                 */
 97 #define PCID_KERNEL     0x000           /* kernel's PCID                */
 98 #define PCID_USER       0x001           /* user-space PCID              */

 92 #define FMT_CR3 "\20\5pcd\4pwt"

100 /* CR4 Register */

102 #define CR4_VME         0x0001          /* virtual-8086 mode extensions */
103 #define CR4_PVI         0x0002          /* protected-mode virtual interrupts */
104 #define CR4_TSD         0x0004          /* time stamp disable           */
105 #define CR4_DE          0x0008          /* debugging extensions         */
106 #define CR4_PSE         0x0010          /* page size extensions         */
107 #define CR4_PAE         0x0020          /* physical address extension   */
108 #define CR4_MCE         0x0040          /* machine check enable         */
109 #define CR4_PGE         0x0080          /* page global enable           */
110 #define CR4_PCE         0x0100          /* perf-monitoring counter enable */
111 #define CR4_OSFXSR      0x0200          /* OS fxsave/fxrstor support    */
112 #define CR4_OSXMMEXCPT  0x0400          /* OS unmasked exception support */
113                                         /* 0x0800 reserved */
114                                         /* 0x1000 reserved */
115 #define CR4_VMXE        0x2000
116 #define CR4_SMXE        0x4000
117 #define CR4_PCIDE       0x20000         /* PCID enable */
```

```
118 #define CR4_OSXSAVE     0x40000         /* OS xsave/xrestore support    */
119 #define CR4_SMEP        0x100000        /* NX for user pages in kernel */
120 #define CR4_SMAP        0x200000        /* kernel can't access user pages */

122 #define FMT_CR4                                         \
123         "\20\26smap\25smep\23osxsav\22pcide"            \
116         "\20\26smap\25smep\23osxsav"                    \
124         "\17smxe\16vmxe\13xmme\12fxsr\11pce\10pge"      \
125         "\7mce\6pae\5pse\4de\3tsd\2pvi\1vme"

127 /*
128  * Enable the SSE-related control bits to explain to the processor that
129  * we're managing XMM state and exceptions
130  */
131 #define CR4_ENABLE_SSE_FLAGS(cr)        \
132         ((cr) | CR4_OSFXSR | CR4_OSXMMEXCPT)

134 /*
135  * Disable the SSE-related control bits to explain to the processor
136  * that we're NOT managing XMM state
137  */
138 #define CR4_DISABLE_SSE_FLAGS(cr)       \
139         ((cr) & ~(uint32_t)(CR4_OSFXSR | CR4_OSXMMEXCPT))

141 /* Intel's SYSENTER configuration registers */

143 #define MSR_INTC_SEP_CS 0x174           /* kernel code selector MSR */
144 #define MSR_INTC_SEP_ESP 0x175          /* kernel esp MSR */
145 #define MSR_INTC_SEP_EIP 0x176          /* kernel eip MSR */

147 /* Intel's microcode registers */
148 #define MSR_INTC_UCODE_WRITE            0x79    /* microcode write */
149 #define MSR_INTC_UCODE_REV              0x8b    /* microcode revision */
150 #define INTC_UCODE_REV_SHIFT            32      /* Bits 63:32 */

152 /* Intel's platform identification */
153 #define MSR_INTC_PLATFORM_ID            0x17
154 #define INTC_PLATFORM_ID_SHIFT          50      /* Bit 52:50 */
155 #define INTC_PLATFORM_ID_MASK           0x7

157 /* AMD's EFER register */

159 #define MSR_AMD_EFER    0xc0000080      /* extended feature enable MSR */

161 #define AMD_EFER_FFXSR  0x4000          /* fast fxsave/fxrstor          */
162 #define AMD_EFER_SVME   0x1000          /* svm enable                   */
163 #define AMD_EFER_NXE    0x0800          /* no-execute enable            */
164 #define AMD_EFER_LMA    0x0400          /* long mode active (read-only) */
165 #define AMD_EFER_LME    0x0100          /* long mode enable             */
166 #define AMD_EFER_SCE    0x0001          /* system call extensions       */

168 #define FMT_AMD_EFER \
169         "\20\17ffxsr\15svme\14nxe\13lma\11lme\1sce"

171 /* AMD's SYSCFG register */

173 #define MSR_AMD_SYSCFG  0xc0000010      /* system configuration MSR */

175 #define AMD_SYSCFG_TOM2 0x200000        /* MtrrTom2En */
176 #define AMD_SYSCFG_MVDM 0x100000        /* MtrrVarDramEn */
177 #define AMD_SYSCFG_MFDM 0x080000        /* MtrrFixDramModEn */
178 #define AMD_SYSCFG_MFDE 0x040000        /* MtrrFixDramEn */

180 #define FMT_AMD_SYSCFG \
181         "\20\26tom2\25mvdm\24mfdm\23mfde"
```



```
183 /* AMD's syscall/sysret MSRs */

185 #define MSR_AMD_STAR    0xc0000081      /* %cs:%ss:%cs:%ss:%eip for syscall */
186 #define MSR_AMD_LSTAR   0xc0000082      /* target %rip of 64-bit syscall */
187 #define MSR_AMD_CSTAR   0xc0000083      /* target %rip of 32-bit syscall */
188 #define MSR_AMD_SFMASK  0xc0000084      /* syscall flag mask */

190 /* AMD's FS.base and GS.base MSRs */

192 #define MSR_AMD_FSBASE  0xc0000100      /* 64-bit base address for %fs */
193 #define MSR_AMD_GSBASE  0xc0000101      /* 64-bit base address for %gs */
194 #define MSR_AMD_KGSBASE 0xc0000102      /* swapgs swaps this with gsbase */
195 #define MSR_AMD_TSCAUX  0xc0000103      /* %ecx value on rdtscp insn */

197 /* AMD's configuration MSRs, weakly documented in the revision guide */

199 #define MSR_AMD_DC_CFG  0xc0011022

201 #define AMD_DC_CFG_DIS_CNV_WC_SSO       (UINT64_C(1) << 3)
202 #define AMD_DC_CFG_DIS_SMC_CHK_BUF      (UINT64_C(1) << 10)

204 /* AMD's HWCR MSR */

206 #define MSR_AMD_HWCR    0xc0010015

208 #define AMD_HWCR_TLBCACHEDIS            (UINT64_C(1) << 3)
209 #define AMD_HWCR_FFDIS                  0x00040 /* disable TLB Flush Filter */
210 #define AMD_HWCR_MCI_STATUS_WREN        0x40000 /* enable write of MCi_STATUS */

212 /* AMD's NorthBridge Config MSR, SHOULD ONLY BE WRITTEN TO BY BIOS */

214 #define MSR_AMD_NB_CFG  0xc001001f

216 #define AMD_NB_CFG_SRQ_HEARTBEAT        (UINT64_C(1) << 20)
217 #define AMD_NB_CFG_SRQ_SPR              (UINT64_C(1) << 32)

219 #define MSR_AMD_BU_CFG  0xc0011023

221 #define AMD_BU_CFG_E298                 (UINT64_C(1) << 1)

223 #define MSR_AMD_DE_CFG  0xc0011029

225 #define AMD_DE_CFG_E721                 (UINT64_C(1))

227 /* AMD's osvw MSRs */
228 #define MSR_AMD_OSVW_ID_LEN             0xc0010140
229 #define MSR_AMD_OSVW_STATUS             0xc0010141


232 #define OSVW_ID_LEN_MASK                0xffffULL
233 #define OSVW_ID_CNT_PER_MSR             64

235 /*
236  * Enable PCI Extended Configuration Space (ECS) on Greyhound
237  */
238 #define AMD_GH_NB_CFG_EN_ECS            (UINT64_C(1) << 46)

240 /* AMD microcode patch loader */
241 #define MSR_AMD_PATCHLEVEL      0x8b
242 #define MSR_AMD_PATCHLOADER     0xc0010020

244 #ifdef __cplusplus
245 }
246 #endif

248 #endif  /* !_SYS_CONTROLREGS_H */
```

```
**********************************************************
    3809 Fri Apr  6 17:25:13 2018
new/usr/src/uts/intel/sys/kdi_machimpl.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
/*
 * 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 _SYS_KDI_MACHIMPL_H
#define _SYS_KDI_MACHIMPL_H

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

/*
 * The Kernel/Debugger interface.  The operations provided by the kdi_t,
 * defined below, comprise the Debugger -> Kernel portion of the interface,
 * and are to be used only when the system has been stopped.
 */

#include <sys/modctl.h>
#include <sys/types.h>
#include <sys/cpuvar.h>
#include <sys/kdi_regs.h>

#ifdef __cplusplus
extern "C" {
#endif

typedef void (*kdi_main_t)(kdi_cpusave_t *);

typedef struct kdi_memrange {
        caddr_t mr_base;
        caddr_t mr_lim;
} kdi_memrange_t;

#define KDI_MEMRANGES_MAX       2

typedef struct kdi_mach {
```



```
        void (*mkdi_activate)(kdi_main_t, kdi_cpusave_t *, uint_t);
        void (*mkdi_deactivate)(void);

        void (*mkdi_idt_switch)(kdi_cpusave_t *);

        void (*mkdi_update_drreg)(kdi_drreg_t *);
        void (*mkdi_set_debug_msrs)(kdi_msr_t *);

        uintptr_t (*mkdi_get_userlimit)(void);

        int (*mkdi_get_cpuinfo)(uint_t *, uint_t *, uint_t *);

        void (*mkdi_stop_slaves)(int, int);

        void (*mkdi_start_slaves)(void);

        void (*mkdi_slave_wait)(void);

        void (*mkdi_memrange_add)(caddr_t, size_t);

        void (*mkdi_reboot)(void);
} kdi_mach_t;

#define mkdi_activate                   kdi_mach.mkdi_activate
#define mkdi_deactivate                 kdi_mach.mkdi_deactivate
#define mkdi_idt_switch                 kdi_mach.mkdi_idt_switch
#define mkdi_update_drreg               kdi_mach.mkdi_update_drreg
#define mkdi_set_debug_msrs             kdi_mach.mkdi_set_debug_msrs
#define mkdi_get_userlimit              kdi_mach.mkdi_get_userlimit
#define mkdi_get_cpuinfo                kdi_mach.mkdi_get_cpuinfo
#define mkdi_stop_slaves                kdi_mach.mkdi_stop_slaves
#define mkdi_start_slaves               kdi_mach.mkdi_start_slaves
#define mkdi_slave_wait                 kdi_mach.mkdi_slave_wait
#define mkdi_memrange_add               kdi_mach.mkdi_memrange_add
#define mkdi_reboot                     kdi_mach.mkdi_reboot

extern void hat_kdi_init(void);

extern ulong_t kdi_getdr0(void), kdi_getdr1(void), kdi_getdr2(void);
extern ulong_t kdi_getdr3(void), kdi_getdr6(void), kdi_getdr7(void);
extern void kdi_setdr0(ulong_t), kdi_setdr1(ulong_t), kdi_setdr2(ulong_t);
extern void kdi_setdr3(ulong_t), kdi_setdr6(ulong_t), kdi_setdr7(ulong_t);
extern ulong_t kdi_dreg_get(int);
extern void kdi_dreg_set(int, ulong_t);
extern void kdi_update_drreg(kdi_drreg_t *);
extern void kdi_set_debug_msrs(kdi_msr_t *);
extern void kdi_cpu_debug_init(kdi_cpusave_t *);

extern void kdi_cpu_init(void);
extern void kdi_xc_others(int, void (*)(void));
extern void kdi_start_slaves(void);
extern void kdi_slave_wait(void);

extern void kdi_idtr_set(gate_desc_t *, size_t);
extern void kdi_idt_write(struct gate_desc *, uint_t);
extern void kdi_idt_sync(void);
extern void kdi_idt_switch(kdi_cpusave_t *);
#ifdef __xpv
extern void kdi_idtr_write(desctbr_t *);
#else
#define kdi_idtr_write(idtr) wr_idtr(idtr)
#endif

extern void kdi_activate(kdi_main_t, kdi_cpusave_t *, uint_t);
extern void kdi_deactivate(void);
extern void kdi_stop_slaves(int, int);
```

```
 119 extern void kdi_memrange_add(caddr_t, size_t);
 120 extern void kdi_reboot(void);

 122 #ifdef __cplusplus
 123 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    3021 Fri Apr  6 17:25:14 2018
new/usr/src/uts/intel/sys/kdi_regs.h
9210 remove KMDB branch debugging support
9211 ::crregs could do with cr2/cr3 support
9209 ::ttrace should be able to filter by thread
Reviewed by: Patrick Mooney <patrick.mooney@joyent.com>
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 _SYS_KDI_REGS_H
#define _SYS_KDI_REGS_H

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

#ifndef _ASM
#include <sys/types.h>
#include <sys/segments.h>
#include <sys/regset.h>
#include <sys/privregs.h>
#endif

#if defined(__amd64)
#include <amd64/sys/kdi_regs.h>
#elif defined(__i386)
#include <ia32/sys/kdi_regs.h>
#endif

#ifdef __cplusplus
extern "C" {
#endif

#define KDI_NCRUMBS     5

#define KDI_CPU_STATE_NONE              0
#define KDI_CPU_STATE_MASTER            1
#define KDI_CPU_STATE_SLAVE             2

#define KDIREG_DRCTL_WPALLEN_MASK       0x000000ff
#define KDIREG_DRSTAT_RESERVED          0xffff0ff0
```



```
#define KDIREG_DRCTL_RESERVED           0x00000700

#define KDI_MSR_READ            0x1     /* read during entry (unlimited) */
#define KDI_MSR_WRITE           0x2     /* write during exit (unlimited) */
#define KDI_MSR_WRITEDELAY      0x4     /* write after last branch (<= 1) */
#define KDI_MSR_CLEARENTRY      0x3     /* clear before 1st branch (<= 1) */

#ifndef _ASM

/*
 * We maintain a ring buffer of bread crumbs for debugging purposes.  The
 * current buffer pointer is advanced along the ring with each intercepted
 * trap (debugger entry, invalid memory access, fault during step, etc).
 */
typedef struct kdi_crumb {
        greg_t krm_cpu_state;   /* This CPU's state at last entry */
        greg_t krm_pc;          /* Instruction pointer at trap */
        greg_t krm_sp;          /* Stack pointer at trap */
        greg_t krm_trapno;      /* The last trap number */
        greg_t krm_flag;        /* KAIF_CRUMB_F_* */
} kdi_crumb_t;
______unchanged_portion_omitted_

typedef struct kdi_msr {
        uint_t          msr_num;
        uint_t          msr_type;
        union {
                uint64_t *_msr_valp;
                uint64_t _msr_val;
        } _u;
} kdi_msr_t;

#define kdi_msr_val     _u._msr_val
#define kdi_msr_valp    _u._msr_valp

/*
 * Data structure used to hold all of the state for a given CPU.
 */
typedef struct kdi_cpusave {
        greg_t                  *krs_gregs;     /* saved registers */

        kdi_drreg_t             krs_dr;         /* saved debug registers */

        user_desc_t             *krs_gdt;       /* GDT address */
        gate_desc_t             *krs_idt;       /* IDT address */

        greg_t                  krs_cr0;        /* saved %cr0 */

        kdi_msr_t               *krs_msr;       /* ptr to MSR save area */

        uint_t                  krs_cpu_state;  /* KDI_CPU_STATE_* mstr/slv */
        uint_t                  krs_cpu_flushed; /* Have caches been flushed? */
        uint_t                  krs_cpu_id;     /* this CPU's ID */

        /* Bread crumb ring buffer */
        ulong_t                 krs_curcrumbidx; /* Current krs_crumbs idx */
        kdi_crumb_t             *krs_curcrumb;  /* Pointer to current crumb */
        kdi_crumb_t             krs_crumbs[KDI_NCRUMBS]; /* Crumbs */
} kdi_cpusave_t;
______unchanged_portion_omitted_
```

```
**********************************************************
   24903 Fri Apr  6 17:25:14 2018
new/usr/src/uts/intel/sys/segments.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
**********************************************************
```

```
/*
 * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright 2018 Joyent, Inc.
 * Copyright 2016 Joyent, Inc.
 */

#ifndef _SYS_SEGMENTS_H
#define _SYS_SEGMENTS_H

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Copyright (c) 1989, 1990 William F. Jolitz
 * Copyright (c) 1990 The Regents of the University of California.
 * All rights reserved.
 *
 * This code is derived from software contributed to Berkeley by
 * William Jolitz.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ''AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 *      from: @(#)segments.h    7.1 (Berkeley) 5/9/91
 * $FreeBSD: src/sys/i386/include/segments.h,v 1.34 2003/09/10 01:07:04
 * jhb Exp $
 *
 * 386 Segmentation Data Structures and definitions
 *      William F. Jolitz (william@ernie.berkeley.edu) 6/20/1989
 */
```



```
#include <sys/tss.h>

/*
 * Selector register format
 * CS, DS, ES, FS, GS, SS
 *
 *  15                  3  2  1 0
 * +---------------------+---+----+
 * |          SI         |TI |RPL |
 * +---------------------+---+----+
 *
 * SI  = selector index
 * TI  = table indicator (0 = GDT, 1 = LDT)
 * RPL = requestor privilege level
 */
#if !defined(_ASM) || defined(__GNUC_AS__)
#define IDXTOSEL(s)     ((s) << 3)              /* index to selector */
#define SEL_GDT(s, r)   (IDXTOSEL(s) | r)       /* global sel */
#else
#define IDXTOSEL(s)     [s << 3]
#define SEL_GDT(s, r)   [IDXTOSEL(s) | r]
#endif

#define SELTOIDX(s)     ((s) >> 3)      /* selector to index */

/*
 * SEL_(KPL,UPL,XPL) is the RPL or DPL value for code and data selectors
 * and their descriptors respectively.
 *
 * TRP_(KPL,UPL,XPL) is used to indicate the DPL for system gates only.
 *
 * This distinction is important to support para-virt guests on the
 * 64-bit hypervisor. Both guest kernel and user run in ring 3 and the
 * hypervisor runs in ring 0. When the kernel creates its trap and
 * interrupt gates it needs a way to prevent any arbitrary int $n
 * instruction from entering a gate that is not expected. The hypervisor
 * allows these gates to have a DPL from 1 to 3. By creating gates
 * with a DPL below user (ring 3) the int $n will generate a #gp fault
 * which the hypervisor catches and forwards to the guest.
 */
#if defined(__xpv)

#if defined(__amd64)

#define SEL_XPL         0               /* hypervisor privilege level */
#define SEL_KPL         3               /* both kernel and user in ring 3 */
#define TRP_KPL         1               /* system gate priv (user blocked) */
#define TRP_XPL         0               /* system gate priv (hypervisor) */

#define IST_DBG         0
#elif defined(__i386)

#define SEL_XPL         0               /* hypervisor privilege level */
#define SEL_KPL         1               /* kernel privilege level */
#define TRP_KPL         SEL_KPL         /* system gate priv (user blocked) */

#endif  /* __i386 */

#define TRP_XPL         0               /* system gate priv (hypervisor) */

#else   /* __xpv */

#define SEL_KPL         0               /* kernel privilege level on metal */
#define TRP_KPL         SEL_KPL         /* system gate priv (user blocked) */
```

```
114 #define IST_DF          1
115 #define IST_NMI         2
116 #define IST_MCE         3
117 #define IST_DBG         4
118 #define IST_NESTABLE    5
119 #define IST_DEFAULT     6

121 #endif  /* __xpv */

123 #define IST_NONE        0

125 #define SEL_UPL         3               /* user priority level */
126 #define TRP_UPL         3               /* system gate priv (user allowed) */
127 #define SEL_TI_LDT      4               /* local descriptor table */
128 #define SEL_LDT(s)      (IDXTOSEL(s) | SEL_TI_LDT | SEL_UPL)    /* local sel */
129 #define CPL_MASK        3               /* RPL mask for selector */
130 #define SELISLDT(s)     (((s) & SEL_TI_LDT) == SEL_TI_LDT)
131 #define SELISUPL(s)     (((s) & CPL_MASK) == SEL_UPL)

133 #ifndef _ASM

135 typedef uint16_t        selector_t;     /* selector reigster */

137 /*
138  * Hardware descriptor table register format for GDT and IDT.
139  */
140 #if defined(__amd64)

142 #pragma pack(2)
143 typedef struct {
144         uint16_t dtr_limit;     /* table limit */
145         uint64_t dtr_base;      /* table base address  */
146 } desctbr_t;
______unchanged_portion_omitted_

384 #define GATESEG_GETOFFSET(sgd)  ((uintptr_t)((sgd)->sgd_looffset |      \
385                                 (sgd)->sgd_hioffset << 16 |             \
386                                 (uint64_t)((sgd)->sgd_hi64offset) << 32))

388 #endif  /* __amd64 */

390 /*
391  * functions for initializing and updating segment descriptors.
392  */
393 #if defined(__amd64)

395 extern void set_usegd(user_desc_t *, uint_t, void *, size_t, uint_t, uint_t,
396     uint_t, uint_t);

398 #elif defined(__i386)

400 extern void set_usegd(user_desc_t *, void *, size_t, uint_t, uint_t,
401     uint_t, uint_t);

403 #endif  /* __i386 */

405 extern uint_t idt_vector_to_ist(uint_t);

407 extern void set_gatesegd(gate_desc_t *, void (*)(void), selector_t,
408     uint_t, uint_t, uint_t);

410 extern void set_syssegd(system_desc_t *, void *, size_t, uint_t, uint_t);

412 extern void *get_ssd_base(system_desc_t *);

414 extern void gdt_update_usegd(uint_t, user_desc_t *);
```



```
416 extern int ldt_update_segd(user_desc_t *, user_desc_t *);

418 #if defined(__xpv)

420 extern int xen_idt_to_trap_info(uint_t, gate_desc_t *, void *);
421 extern void xen_idt_write(gate_desc_t *, uint_t);

423 #endif  /* __xen */

425 void init_boot_gdt(user_desc_t *);

427 #endif  /* _ASM */

429 /*
430  * Common segment parameter defintions for granularity, default
431  * operand size and operaton mode.
432  */
433 #define SDP_BYTES       0       /* segment limit scaled to bytes */
434 #define SDP_PAGES       1       /* segment limit scaled to pages */
435 #define SDP_OP32        1       /* code and data default operand = 32 bits */
436 #define SDP_LONG        1       /* long mode code segment (64 bits) */
437 #define SDP_SHORT       0       /* compat/legacy code segment (32 bits) */
438 /*
439  * System segments and gate types.
440  *
441  * In long mode i386 32-bit ldt, tss, call, interrupt and trap gate
442  * types are redefined into 64-bit equivalents.
443  */
444 #define SDT_SYSNULL      0      /* system null */
445 #define SDT_SYS286TSS    1      /* system 286 TSS available */
446 #define SDT_SYSLDT       2      /* system local descriptor table */
447 #define SDT_SYS286BSY    3      /* system 286 TSS busy */
448 #define SDT_SYS286CGT    4      /* system 286 call gate */
449 #define SDT_SYSTASKGT    5      /* system task gate */
450 #define SDT_SYS286IGT    6      /* system 286 interrupt gate */
451 #define SDT_SYS286TGT    7      /* system 286 trap gate */
452 #define SDT_SYSNULL2     8      /* system null again */
453 #define SDT_SYSTSS       9      /* system TSS available */
454 #define SDT_SYSNULL3    10      /* system null again */
455 #define SDT_SYSTSSBSY   11      /* system TSS busy */
456 #define SDT_SYSCGT      12      /* system call gate */
457 #define SDT_SYSNULL4    13      /* system null again */
458 #define SDT_SYSIGT      14      /* system interrupt gate */
459 #define SDT_SYSTGT      15      /* system trap gate */

461 /*
462  * Memory segment types.
463  *
464  * While in long mode expand-down, writable and accessed type field
465  * attributes are ignored. Only the conforming bit is loaded by hardware
466  * for long mode code segment descriptors.
467  */
468 #define SDT_MEMRO       16      /* read only */
469 #define SDT_MEMROA      17      /* read only accessed */
470 #define SDT_MEMRW       18      /* read write */
471 #define SDT_MEMRWA      19      /* read write accessed */
472 #define SDT_MEMROD      20      /* read only expand dwn limit */
473 #define SDT_MEMRODA     21      /* read only expand dwn limit accessed */
474 #define SDT_MEMRWD      22      /* read write expand dwn limit */
475 #define SDT_MEMRWDA     23      /* read write expand dwn limit accessed */
476 #define SDT_MEME        24      /* execute only */
477 #define SDT_MEMEA       25      /* execute only accessed */
478 #define SDT_MEMER       26      /* execute read */
479 #define SDT_MEMERA      27      /* execute read accessed */
480 #define SDT_MEMEC       28      /* execute only conforming */
```

```
481 #define SDT_MEMEAC      29      /* execute only accessed conforming */
482 #define SDT_MEMERC      30      /* execute read conforming */
483 #define SDT_MEMERAC     31      /* execute read accessed conforming */

485 /*
486  * Entries in the Interrupt Descriptor Table (IDT)
487  */
488 #define IDT_DE          0       /* #DE: Divide Error */
489 #define IDT_DB          1       /* #DB: Debug */
490 #define IDT_NMI         2       /* Nonmaskable External Interrupt */
491 #define IDT_BP          3       /* #BP: Breakpoint */
492 #define IDT_OF          4       /* #OF: Overflow */
493 #define IDT_BR          5       /* #BR: Bound Range Exceeded */
494 #define IDT_UD          6       /* #UD: Undefined/Invalid Opcode */
495 #define IDT_NM          7       /* #NM: No Math Coprocessor */
496 #define IDT_DF          8       /* #DF: Double Fault */
497 #define IDT_FPUGP       9       /* Coprocessor Segment Overrun */
498 #define IDT_TS          10      /* #TS: Invalid TSS */
499 #define IDT_NP          11      /* #NP: Segment Not Present */
500 #define IDT_SS          12      /* #SS: Stack Segment Fault */
501 #define IDT_GP          13      /* #GP: General Protection Fault */
502 #define IDT_PF          14      /* #PF: Page Fault */
503 #define IDT_MF          16      /* #MF: FPU Floating-Point Error */
504 #define IDT_AC          17      /* #AC: Alignment Check */
505 #define IDT_MC          18      /* #MC: Machine Check */
506 #define IDT_XF          19      /* #XF: SIMD Floating-Point Exception */
507 #define NIDT            256     /* size in entries of IDT */

509 /*
510  * Entries in the Global Descriptor Table (GDT)
511  *
512  * We make sure to space the system descriptors (LDT's, TSS')
513  * such that they are double gdt slot aligned. This is because
514  * in long mode system segment decriptors expand to 128 bits.
515  *
516  * GDT_LWPFS and GDT_LWPGS must be the same for both 32 and 64-bit
517  * kernels. See setup_context in libc. 64-bit processes must set
518  * %fs or %gs to null selector to use 64-bit fsbase or gsbase
519  * respectively.
520  */
521 #define GDT_NULL        0       /* null */
522 #define GDT_B32DATA     1       /* dboot 32 bit data descriptor */
523 #define GDT_B32CODE     2       /* dboot 32 bit code descriptor */
524 #define GDT_B16CODE     3       /* bios call 16 bit code descriptor */
525 #define GDT_B16DATA     4       /* bios call 16 bit data descriptor */
526 #define GDT_B64CODE     5       /* dboot 64 bit code descriptor */
527 #define GDT_BGSTMP      7       /* kmdb descriptor only used early in boot */
528 #define GDT_CPUID       16      /* store numeric id of current CPU */

530 #if defined(__amd64)

532 #define GDT_KCODE       6       /* kernel code seg %cs */
533 #define GDT_KDATA       7       /* kernel data seg %ds */
534 #define GDT_U32CODE     8       /* 32-bit process on 64-bit kernel %cs */
535 #define GDT_UDATA       9       /* user data seg %ds (32 and 64 bit) */
536 #define GDT_UCODE       10      /* native user code  seg %cs */
537 #define GDT_LDT         12      /* (12-13) LDT for current process */
538 #define GDT_KTSS        14      /* (14-15) kernel tss */
539 #define GDT_FS          GDT_NULL /* kernel %fs segment selector */
540 #define GDT_GS          GDT_NULL /* kernel %gs segment selector */
541 #define GDT_LWPFS       55      /* lwp private %fs segment selector (32-bit) */
542 #define GDT_LWPGS       56      /* lwp private %gs segment selector (32-bit) */
543 #define GDT_BRANDMIN    57      /* first entry in GDT for brand usage */
544 #define GDT_BRANDMAX    61      /* last entry in GDT for brand usage */
545 #define NGDT            62      /* number of entries in GDT */
```



```
547 /*
548  * This selector is only used in the temporary GDT used to bring additional
549  * CPUs from 16-bit real mode into long mode in real_mode_start().
550  */
551 #define TEMPGDT_KCODE64 1       /* 64-bit code selector */

553 #elif defined(__i386)

555 #define GDT_LDT         40      /* LDT for current process */
556 #define GDT_KTSS        42      /* kernel tss */
557 #define GDT_KCODE       43      /* kernel code seg %cs */
558 #define GDT_KDATA       44      /* kernel data seg %ds */
559 #define GDT_UCODE       45      /* native user code  seg %cs */
560 #define GDT_UDATA       46      /* user data seg %ds (32 and 64 bit) */
561 #define GDT_DBFLT       47      /* double fault #DF selector */
562 #define GDT_FS          53      /* kernel %fs segment selector */
563 #define GDT_GS          54      /* kernel %gs segment selector */
564 #define GDT_LWPFS       55      /* lwp private %fs segment selector */
565 #define GDT_LWPGS       56      /* lwp private %gs segment selector */
566 #define GDT_BRANDMIN    57      /* first entry in GDT for brand usage */
567 #define GDT_BRANDMAX    61      /* last entry in GDT for brand usage */
568 #if !defined(__xpv)
569 #define NGDT            90      /* number of entries in GDT */
570 #else
571 #define NGDT            512     /* single 4K page for the hypervisor */
572 #endif

574 #endif  /* __i386 */

576 /*
577  * Convenient selector definitions.
578  */

580 /*
581  * XXPV 64 bit Xen only allows the guest %cs/%ss be the private ones it
582  * provides, not the ones we create for ourselves.  See FLAT_RING3_CS64 in
583  * public/arch-x86_64.h
584  *
585  * 64-bit Xen runs paravirtual guests in ring 3 but emulates them running in
586  * ring 0 by clearing CPL in %cs value pushed on guest exception stacks.
587  * Therefore we will have KCS_SEL value indicate ring 0 and use that everywhere
588  * in the kernel. But in the few files where we initialize segment registers or
589  * create and update descriptors we will explicity OR in SEL_KPL (ring 3) for
590  * kernel %cs. See desctbls.c for an example.
591  */

593 #if defined(__xpv) && defined(__amd64)
594 #define KCS_SEL         0xe030          /* FLAT_RING3_CS64 & 0xFFF0 */
595 #define KDS_SEL         0xe02b          /* FLAT_RING3_SS64 */
596 #else
597 #define KCS_SEL         SEL_GDT(GDT_KCODE, SEL_KPL)
598 #define KDS_SEL         SEL_GDT(GDT_KDATA, SEL_KPL)
599 #endif

601 #define UCS_SEL         SEL_GDT(GDT_UCODE, SEL_UPL)
602 #if defined(__amd64)
603 #define TEMP_CS64_SEL   SEL_GDT(TEMPGDT_KCODE64, SEL_KPL)
604 #define U32CS_SEL       SEL_GDT(GDT_U32CODE, SEL_UPL)
605 #endif

607 #define UDS_SEL         SEL_GDT(GDT_UDATA, SEL_UPL)
608 #define ULDT_SEL        SEL_GDT(GDT_LDT, SEL_KPL)
609 #define KTSS_SEL        SEL_GDT(GDT_KTSS, SEL_KPL)
610 #define DFTSS_SEL       SEL_GDT(GDT_DBFLT, SEL_KPL)
611 #define KFS_SEL         0
612 #define KGS_SEL         SEL_GDT(GDT_GS, SEL_KPL)
```

```
613 #define LWPFS_SEL       SEL_GDT(GDT_LWPFS, SEL_UPL)
614 #define LWPGS_SEL       SEL_GDT(GDT_LWPGS, SEL_UPL)
615 #define BRANDMIN_SEL    SEL_GDT(GDT_BRANDMIN, SEL_UPL)
616 #define BRANDMAX_SEL    SEL_GDT(GDT_BRANDMAX, SEL_UPL)

618 #define B64CODE_SEL     SEL_GDT(GDT_B64CODE, SEL_KPL)
619 #define B32CODE_SEL     SEL_GDT(GDT_B32CODE, SEL_KPL)
620 #define B32DATA_SEL     SEL_GDT(GDT_B32DATA, SEL_KPL)
621 #define B16CODE_SEL     SEL_GDT(GDT_B16CODE, SEL_KPL)
622 #define B16DATA_SEL     SEL_GDT(GDT_B16DATA, SEL_KPL)

624 /*
625  * Temporary %gs descriptor used by kmdb with -d option. Only lives
626  * in boot's GDT and is not copied into kernel's GDT from boot.
627  */
628 #define KMDBGS_SEL      SEL_GDT(GDT_BGSTMP, SEL_KPL)

630 /*
631  * Selector used for kdi_idt when kmdb has taken over the IDT.
632  */
633 #if defined(__amd64)
634 #define KMDBCODE_SEL    B64CODE_SEL
635 #else
636 #define KMDBCODE_SEL    B32CODE_SEL
637 #endif

639 /*
640  * Entries in default Local Descriptor Table (LDT) for every process.
641  */
642 #define LDT_SYSCALL     0       /* call gate for libc.a (obsolete) */
643 #define LDT_SIGCALL     1       /* EOL me, call gate for static sigreturn */
644 #define LDT_RESVD1      2       /* old user %cs */
645 #define LDT_RESVD2      3       /* old user %ds */
646 #define LDT_ALTSYSCALL  4       /* alternate call gate for system calls */
647 #define LDT_ALTSIGCALL  5       /* EOL me, alternate call gate for sigreturn */
648 #define LDT_UDBASE      6       /* user descriptor base index */
649 #define MINNLDT         512     /* Current min solaris ldt size (1 4K page) */
650 #define MAXNLDT         8192    /* max solaris ldt size (16 4K pages) */

652 #ifdef _KERNEL
653 #define LDT_CPU_SIZE    (16 * 4096)     /* Size of kernel per-CPU allocation */
654 #endif

656 #ifndef _ASM

658 extern  gate_desc_t     *idt0;
659 extern  desctbr_t       idt0_default_reg;
660 extern  user_desc_t     *gdt0;

662 extern user_desc_t      zero_udesc;
663 extern user_desc_t      null_udesc;
664 extern system_desc_t    null_sdesc;

666 #if defined(__amd64)
667 extern user_desc_t      zero_u32desc;
668 #endif
669 #if defined(__amd64)
670 extern user_desc_t      ucs_on;
671 extern user_desc_t      ucs_off;
672 extern user_desc_t      ucs32_on;
673 extern user_desc_t      ucs32_off;
674 #endif  /* __amd64 */

676 extern tss_t *ktss0;

678 #if defined(__i386)
```



```
679 extern tss_t *dftss0;
680 #endif  /* __i386 */

682 extern void div0trap(), dbgtrap(), nmiint(), brktrap(), ovflotrap();
683 extern void boundstrap(), invoptrap(), ndptrap();
684 #if !defined(__xpv)
685 extern void syserrtrap();
686 #endif
687 extern void invaltrap(), invtsstrap(), segnptrap(), stktrap();
688 extern void gptrap(), pftrap(), ndperr();
689 extern void overrun(), resvtrap();
690 extern void _start(), cmnint();
691 extern void achktrap(), mcetrap();
692 extern void xmtrap();
693 extern void fasttrap();
694 extern void dtrace_ret();

696 /* KPTI trampolines */
697 extern void tr_invaltrap();
698 extern void tr_div0trap(), tr_dbgtrap(), tr_nmiint(), tr_brktrap();
699 extern void tr_ovflotrap(), tr_boundstrap(), tr_invoptrap(), tr_ndptrap();
700 #if !defined(__xpv)
701 extern void tr_syserrtrap();
702 #endif
703 extern void tr_invaltrap(), tr_invtsstrap(), tr_segnptrap(), tr_stktrap();
704 extern void tr_gptrap(), tr_pftrap(), tr_ndperr();
705 extern void tr_overrun(), tr_resvtrap();
706 extern void tr_achktrap(), tr_mcetrap();
707 extern void tr_xmtrap();
708 extern void tr_fasttrap();
709 extern void tr_dtrace_ret();

711 #if !defined(__amd64)
712 extern void pentium_pftrap();
713 #endif

715 extern uint64_t kpti_enable;

717 #endif /* _ASM */

719 #ifdef  __cplusplus
720 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   32559 Fri Apr  6 17:25:14 2018
new/usr/src/uts/intel/sys/x86_archext.h
8956 Implement KPTI
Reviewed by: Jerry Jelinek <jerry.jelinek@joyent.com>
Reviewed by: Robert Mustacchi <rm@joyent.com>
9215 update CPUID defines
Reviewed by: Yuri Pankov <yuripv@yuripv.net>
**********************************************************
```

```
/*
 * 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 (c) 1995, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2011 by Delphix. All rights reserved.
 */
/*
 * Copyright (c) 2010, Intel Corporation.
 * All rights reserved.
 */
/*
 * Copyright 2018 Joyent, Inc.
 * Copyright 2017 Joyent, Inc.
 * Copyright 2012 Jens Elkner <jel+illumos@cs.uni-magdeburg.de>
 * Copyright 2012 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
 * Copyright 2014 Josef 'Jeff' Sipek <jeffpc@josefsipek.net>
 * Copyright 2018 Nexenta Systems, Inc.
 */

#ifndef _SYS_X86_ARCHEXT_H
#define _SYS_X86_ARCHEXT_H

#if !defined(_ASM)
#include <sys/regset.h>
#include <sys/processor.h>
#include <vm/seg_enum.h>
#include <vm/page.h>
#endif  /* _ASM */

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * cpuid instruction feature flags in %edx (standard function 1)
 */

#define CPUID_INTC_EDX_FPU      0x00000001      /* x87 fpu present */
#define CPUID_INTC_EDX_VME      0x00000002      /* virtual-8086 extension */
```



```
#define CPUID_INTC_EDX_DE       0x00000004      /* debugging extensions */
#define CPUID_INTC_EDX_PSE      0x00000008      /* page size extension */
#define CPUID_INTC_EDX_TSC      0x00000010      /* time stamp counter */
#define CPUID_INTC_EDX_MSR      0x00000020      /* rdmsr and wrmsr */
#define CPUID_INTC_EDX_PAE      0x00000040      /* physical addr extension */
#define CPUID_INTC_EDX_MCE      0x00000080      /* machine check exception */
#define CPUID_INTC_EDX_CX8      0x00000100      /* cmpxchg8b instruction */
#define CPUID_INTC_EDX_APIC     0x00000200      /* local APIC */
                                                /* 0x400 - reserved */
#define CPUID_INTC_EDX_SEP      0x00000800      /* sysenter and sysexit */
#define CPUID_INTC_EDX_MTRR     0x00001000      /* memory type range reg */
#define CPUID_INTC_EDX_PGE      0x00002000      /* page global enable */
#define CPUID_INTC_EDX_MCA      0x00004000      /* machine check arch */
#define CPUID_INTC_EDX_CMOV     0x00008000      /* conditional move insns */
#define CPUID_INTC_EDX_PAT      0x00010000      /* page attribute table */
#define CPUID_INTC_EDX_PSE36    0x00020000      /* 36-bit pagesize extension */
#define CPUID_INTC_EDX_PSN      0x00040000      /* processor serial number */
#define CPUID_INTC_EDX_CLFSH    0x00080000      /* clflush instruction */
                                                /* 0x100000 - reserved */
#define CPUID_INTC_EDX_DS       0x00200000      /* debug store exists */
#define CPUID_INTC_EDX_ACPI     0x00400000      /* monitoring + clock ctrl */
#define CPUID_INTC_EDX_MMX      0x00800000      /* MMX instructions */
#define CPUID_INTC_EDX_FXSR     0x01000000      /* fxsave and fxrstor */
#define CPUID_INTC_EDX_SSE      0x02000000      /* streaming SIMD extensions */
#define CPUID_INTC_EDX_SSE2     0x04000000      /* SSE extensions */
#define CPUID_INTC_EDX_SS       0x08000000      /* self-snoop */
#define CPUID_INTC_EDX_HTT      0x10000000      /* Hyper Thread Technology */
#define CPUID_INTC_EDX_TM       0x20000000      /* thermal monitoring */
#define CPUID_INTC_EDX_IA64     0x40000000      /* Itanium emulating IA32 */
#define CPUID_INTC_EDX_PBE      0x80000000      /* Pending Break Enable */

/*
 * cpuid instruction feature flags in %ecx (standard function 1)
 */

#define CPUID_INTC_ECX_SSE3     0x00000001      /* Yet more SSE extensions */
#define CPUID_INTC_ECX_PCLMULQDQ 0x00000002     /* PCLMULQDQ insn */
#define CPUID_INTC_ECX_DTES64   0x00000004      /* 64-bit DS area */
                                                /* 0x00000004 - reserved */
#define CPUID_INTC_ECX_MON      0x00000008      /* MONITOR/MWAIT */
#define CPUID_INTC_ECX_DSCPL    0x00000010      /* CPL-qualified debug store */
#define CPUID_INTC_ECX_VMX      0x00000020      /* Hardware VM extensions */
#define CPUID_INTC_ECX_SMX      0x00000040      /* Secure mode extensions */
#define CPUID_INTC_ECX_EST      0x00000080      /* enhanced SpeedStep */
#define CPUID_INTC_ECX_TM2      0x00000100      /* thermal monitoring */
#define CPUID_INTC_ECX_SSSE3    0x00000200      /* Supplemental SSE3 insns */
#define CPUID_INTC_ECX_CID      0x00000400      /* L1 context ID */
                                                /* 0x00000800 - reserved */
#define CPUID_INTC_ECX_FMA      0x00001000      /* Fused Multiply Add */
#define CPUID_INTC_ECX_CX16     0x00002000      /* cmpxchg16 */
#define CPUID_INTC_ECX_ETPRD    0x00004000      /* extended task pri messages */
#define CPUID_INTC_ECX_PDCM     0x00008000      /* Perf/Debug Capability MSR */
                                                /* 0x00008000 - reserved */
                                                /* 0x00010000 - reserved */
#define CPUID_INTC_ECX_PCID     0x00020000      /* process-context ids */
                                                /* 0x00020000 - reserved */
#define CPUID_INTC_ECX_DCA      0x00040000      /* direct cache access */
#define CPUID_INTC_ECX_SSE4_1   0x00080000      /* SSE4.1 insns */
#define CPUID_INTC_ECX_SSE4_2   0x00100000      /* SSE4.2 insns */
#define CPUID_INTC_ECX_X2APIC   0x00200000      /* x2APIC */
#define CPUID_INTC_ECX_MOVBE    0x00400000      /* MOVBE insn */
#define CPUID_INTC_ECX_POPCNT   0x00800000      /* POPCNT insn */
#define CPUID_INTC_ECX_TSCDL    0x01000000      /* Deadline TSC */
#define CPUID_INTC_ECX_AES      0x02000000      /* AES insns */
#define CPUID_INTC_ECX_XSAVE    0x04000000      /* XSAVE/XRESTOR insns */
#define CPUID_INTC_ECX_OSXSAVE  0x08000000      /* OS supports XSAVE insns */
```

```
120 #define CPUID_INTC_ECX_AVX      0x10000000      /* AVX supported */
121 #define CPUID_INTC_ECX_F16C     0x20000000      /* F16C supported */
122 #define CPUID_INTC_ECX_RDRAND   0x40000000      /* RDRAND supported */
123 #define CPUID_INTC_ECX_HV       0x80000000      /* Hypervisor */

125 /*
126  * cpuid instruction feature flags in %edx (extended function 0x80000001)
127  */

129 #define CPUID_AMD_EDX_FPU       0x00000001      /* x87 fpu present */
130 #define CPUID_AMD_EDX_VME       0x00000002      /* virtual-8086 extension */
131 #define CPUID_AMD_EDX_DE        0x00000004      /* debugging extensions */
132 #define CPUID_AMD_EDX_PSE       0x00000008      /* page size extensions */
133 #define CPUID_AMD_EDX_TSC       0x00000010      /* time stamp counter */
134 #define CPUID_AMD_EDX_MSR       0x00000020      /* rdmsr and wrmsr */
135 #define CPUID_AMD_EDX_PAE       0x00000040      /* physical addr extension */
136 #define CPUID_AMD_EDX_MCE       0x00000080      /* machine check exception */
137 #define CPUID_AMD_EDX_CX8       0x00000100      /* cmpxchg8b instruction */
138 #define CPUID_AMD_EDX_APIC      0x00000200      /* local APIC */
139                                                 /* 0x00000400 - sysc on K6m6 */
140 #define CPUID_AMD_EDX_SYSC      0x00000800      /* AMD: syscall and sysret */
141 #define CPUID_AMD_EDX_MTRR      0x00001000      /* memory type and range reg */
142 #define CPUID_AMD_EDX_PGE       0x00002000      /* page global enable */
143 #define CPUID_AMD_EDX_MCA       0x00004000      /* machine check arch */
144 #define CPUID_AMD_EDX_CMOV      0x00008000      /* conditional move insns */
145 #define CPUID_AMD_EDX_PAT       0x00010000      /* K7: page attribute table */
146 #define CPUID_AMD_EDX_FCMOV     0x00010000      /* FCMOVcc etc. */
147 #define CPUID_AMD_EDX_PSE36     0x00020000      /* 36-bit pagesize extension */
148                                 /* 0x00040000 - reserved */
149                                 /* 0x00080000 - reserved */
150 #define CPUID_AMD_EDX_NX        0x00100000      /* AMD: no-execute page prot */
151                                 /* 0x00200000 - reserved */
152 #define CPUID_AMD_EDX_MMXamd    0x00400000      /* AMD: MMX extensions */
153 #define CPUID_AMD_EDX_MMX       0x00800000      /* MMX instructions */
154 #define CPUID_AMD_EDX_FXSR      0x01000000      /* fxsave and fxrstor */
155 #define CPUID_AMD_EDX_FFXSR     0x02000000      /* fast fxsave/fxrstor */
156 #define CPUID_AMD_EDX_1GPG      0x04000000      /* 1GB page */
157 #define CPUID_AMD_EDX_TSCP      0x08000000      /* rdtscp instruction */
158                                 /* 0x10000000 - reserved */
159 #define CPUID_AMD_EDX_LM        0x20000000      /* AMD: long mode */
160 #define CPUID_AMD_EDX_3DNowx    0x40000000      /* AMD: extensions to 3DNow! */
161 #define CPUID_AMD_EDX_3DNow     0x80000000      /* AMD: 3DNow! instructions */

163 #define CPUID_AMD_ECX_AHF64     0x00000001      /* LAHF and SAHF in long mode */
164 #define CPUID_AMD_ECX_CMP_LGCY  0x00000002      /* AMD: multicore chip */
165 #define CPUID_AMD_ECX_SVM       0x00000004      /* AMD: secure VM */
166 #define CPUID_AMD_ECX_EAS       0x00000008      /* extended apic space */
167 #define CPUID_AMD_ECX_CR8D      0x00000010      /* AMD: 32-bit mov %cr8 */
168 #define CPUID_AMD_ECX_LZCNT     0x00000020      /* AMD: LZCNT insn */
169 #define CPUID_AMD_ECX_SSE4A     0x00000040      /* AMD: SSE4A insns */
170 #define CPUID_AMD_ECX_MAS       0x00000080      /* AMD: MisAlignSse mnode */
171 #define CPUID_AMD_ECX_3DNP      0x00000100      /* AMD: 3DNowPrefectch */
172 #define CPUID_AMD_ECX_OSVW      0x00000200      /* AMD: OSVW */
173 #define CPUID_AMD_ECX_IBS       0x00000400      /* AMD: IBS */
174 #define CPUID_AMD_ECX_SSE5      0x00000800      /* AMD: Extended AVX */
173 #define CPUID_AMD_ECX_SSE5      0x00000800      /* AMD: SSE5 */
175 #define CPUID_AMD_ECX_SKINIT    0x00001000      /* AMD: SKINIT */
176 #define CPUID_AMD_ECX_WDT       0x00002000      /* AMD: WDT */
177                                 /* 0x00004000 - reserved */
178 #define CPUID_AMD_ECX_LWP       0x00008000      /* AMD: Lightweight profiling */
179 #define CPUID_AMD_ECX_FMA4      0x00010000      /* AMD: 4-operand FMA support */
180                                 /* 0x00020000 - reserved */
181                                 /* 0x00040000 - reserved */
182 #define CPUID_AMD_ECX_NIDMSR    0x00080000      /* AMD: Node ID MSR */
183                                 /* 0x00100000 - reserved */
184 #define CPUID_AMD_ECX_TBM       0x00200000      /* AMD: trailing bit manips. */
```



```
185 #define CPUID_AMD_ECX_TOPOEXT   0x00400000      /* AMD: Topology Extensions */

187 /*
188  * AMD uses %ebx for some of their features (extended function 0x80000008).
189  */
190 #define CPUID_AMD_EBX_ERR_PTR_ZERO      0x00000004 /* AMD: FP Err. Ptr. Zero */

192 /*
193  * Intel now seems to have claimed part of the "extended" function
194  * space that we previously for non-Intel implementors to use.
195  * More excitingly still, they've claimed bit 20 to mean LAHF/SAHF
196  * is available in long mode i.e. what AMD indicate using bit 0.
197  * On the other hand, everything else is labelled as reserved.
198  */
199 #define CPUID_INTC_ECX_AHF64    0x00100000      /* LAHF and SAHF in long mode */

201 /*
202  * Intel also uses cpuid leaf 7 to have additional instructions and features.
203  * Like some other leaves, but unlike the current ones we care about, it
204  * requires us to specify both a leaf in %eax and a sub-leaf in %ecx. To deal
205  * with the potential use of additional sub-leaves in the future, we now
206  * specifically label the EBX features with their leaf and sub-leaf.
207  */
208 #define CPUID_INTC_EBX_7_0_BMI1         0x00000008      /* BMI1 instrs */
209 #define CPUID_INTC_EBX_7_0_HLE          0x00000010      /* HLE */
210 #define CPUID_INTC_EBX_7_0_AVX2         0x00000020      /* AVX2 supported */
211 #define CPUID_INTC_EBX_7_0_SMEP         0x00000080      /* SMEP in CR4 */
212 #define CPUID_INTC_EBX_7_0_BMI2         0x00000100      /* BMI2 instrs */
213 #define CPUID_INTC_EBX_7_0_INVPCID      0x00000400      /* invpcid instr */
214 #define CPUID_INTC_EBX_7_0_MPX          0x00004000      /* Mem. Prot. Ext. */
215 #define CPUID_INTC_EBX_7_0_AVX512F      0x00010000      /* AVX512 foundation */
216 #define CPUID_INTC_EBX_7_0_AVX512DQ     0x00020000      /* AVX512DQ */
217 #define CPUID_INTC_EBX_7_0_RDSEED       0x00040000      /* RDSEED instr */
218 #define CPUID_INTC_EBX_7_0_ADX          0x00080000      /* ADX instrs */
219 #define CPUID_INTC_EBX_7_0_SMAP         0x00100000      /* SMAP in CR 4 */
220 #define CPUID_INTC_EBX_7_0_AVX512IFMA   0x00200000      /* AVX512IFMA */
221 #define CPUID_INTC_EBX_7_0_CLWB         0x01000000      /* CLWB */
222 #define CPUID_INTC_EBX_7_0_AVX512PF     0x04000000      /* AVX512PF */
223 #define CPUID_INTC_EBX_7_0_AVX512ER     0x08000000      /* AVX512ER */
224 #define CPUID_INTC_EBX_7_0_AVX512CD     0x10000000      /* AVX512CD */
225 #define CPUID_INTC_EBX_7_0_SHA          0x20000000      /* SHA extensions */
226 #define CPUID_INTC_EBX_7_0_AVX512BW     0x40000000      /* AVX512BW */
227 #define CPUID_INTC_EBX_7_0_AVX512VL     0x80000000      /* AVX512VL */

229 #define CPUID_INTC_EBX_7_0_ALL_AVX512 \
230         (CPUID_INTC_EBX_7_0_AVX512F | CPUID_INTC_EBX_7_0_AVX512DQ | \
231         CPUID_INTC_EBX_7_0_AVX512IFMA | CPUID_INTC_EBX_7_0_AVX512PF | \
232         CPUID_INTC_EBX_7_0_AVX512ER | CPUID_INTC_EBX_7_0_AVX512CD | \
233         CPUID_INTC_EBX_7_0_AVX512BW | CPUID_INTC_EBX_7_0_AVX512VL)

235 #define CPUID_INTC_ECX_7_0_AVX512VBMI   0x00000002      /* AVX512VBMI */
236 #define CPUID_INTC_ECX_7_0_UMIP         0x00000004      /* UMIP */
237 #define CPUID_INTC_ECX_7_0_PKU          0x00000008      /* umode prot. keys */
238 #define CPUID_INTC_ECX_7_0_OSPKE        0x00000010      /* OSPKE */
239 #define CPUID_INTC_ECX_7_0_AVX512VPOPCDQ 0x00004000     /* AVX512 VPOPCNTDQ */

241 #define CPUID_INTC_ECX_7_0_ALL_AVX512 \
242         (CPUID_INTC_ECX_7_0_AVX512VBMI | CPUID_INTC_ECX_7_0_AVX512VPOPCDQ)

244 #define CPUID_INTC_EDX_7_0_AVX5124NNIW  0x00000004      /* AVX512 4NNIW */
245 #define CPUID_INTC_EDX_7_0_AVX5124FMAPS 0x00000008      /* AVX512 4FMAPS */

247 #define CPUID_INTC_EDX_7_0_ALL_AVX512 \
248         (CPUID_INTC_EDX_7_0_AVX5124NNIW | CPUID_INTC_EDX_7_0_AVX5124FMAPS)

250 /*
```

```
 251  * Intel also uses cpuid leaf 0xd to report additional instructions and features
 252  * when the sub-leaf in %ecx == 1. We label these using the same convention as
 253  * with leaf 7.
 254  */
 255 #define CPUID_INTC_EAX_D_1_XSAVEOPT     0x00000001      /* xsaveopt inst. */
 256 #define CPUID_INTC_EAX_D_1_XSAVEC       0x00000002      /* xsavec inst. */
 257 #define CPUID_INTC_EAX_D_1_XSAVES       0x00000008      /* xsaves inst. */

 259 #define REG_PAT                 0x277
 260 #define REG_TSC                 0x10    /* timestamp counter */
 261 #define REG_APIC_BASE_MSR       0x1b
 262 #define REG_X2APIC_BASE_MSR     0x800   /* The MSR address offset of x2APIC */

 264 #if !defined(__xpv)
 265 /*
 266  * AMD C1E
 267  */
 268 #define MSR_AMD_INT_PENDING_CMP_HALT    0xC0010055
 269 #define AMD_ACTONCMPHALT_SHIFT  27
 270 #define AMD_ACTONCMPHALT_MASK   3
 271 #endif

 273 #define MSR_DEBUGCTL            0x1d9

 275 #define DEBUGCTL_LBR            0x01
 276 #define DEBUGCTL_BTF            0x02

 278 /* Intel P6, AMD */
 279 #define MSR_LBR_FROM            0x1db
 280 #define MSR_LBR_TO              0x1dc
 281 #define MSR_LEX_FROM            0x1dd
 282 #define MSR_LEX_TO              0x1de

 284 /* Intel P4 (pre-Prescott, non P4 M) */
 285 #define MSR_P4_LBSTK_TOS        0x1da
 286 #define MSR_P4_LBSTK_0          0x1db
 287 #define MSR_P4_LBSTK_1          0x1dc
 288 #define MSR_P4_LBSTK_2          0x1dd
 289 #define MSR_P4_LBSTK_3          0x1de

 291 /* Intel Pentium M */
 292 #define MSR_P6M_LBSTK_TOS       0x1c9
 293 #define MSR_P6M_LBSTK_0         0x040
 294 #define MSR_P6M_LBSTK_1         0x041
 295 #define MSR_P6M_LBSTK_2         0x042
 296 #define MSR_P6M_LBSTK_3         0x043
 297 #define MSR_P6M_LBSTK_4         0x044
 298 #define MSR_P6M_LBSTK_5         0x045
 299 #define MSR_P6M_LBSTK_6         0x046
 300 #define MSR_P6M_LBSTK_7         0x047

 302 /* Intel P4 (Prescott) */
 303 #define MSR_PRP4_LBSTK_TOS      0x1da
 304 #define MSR_PRP4_LBSTK_FROM_0   0x680
 305 #define MSR_PRP4_LBSTK_FROM_1   0x681
 306 #define MSR_PRP4_LBSTK_FROM_2   0x682
 307 #define MSR_PRP4_LBSTK_FROM_3   0x683
 308 #define MSR_PRP4_LBSTK_FROM_4   0x684
 309 #define MSR_PRP4_LBSTK_FROM_5   0x685
 310 #define MSR_PRP4_LBSTK_FROM_6   0x686
 311 #define MSR_PRP4_LBSTK_FROM_7   0x687
 312 #define MSR_PRP4_LBSTK_FROM_8   0x688
 313 #define MSR_PRP4_LBSTK_FROM_9   0x689
 314 #define MSR_PRP4_LBSTK_FROM_10  0x68a
 315 #define MSR_PRP4_LBSTK_FROM_11  0x68b
 316 #define MSR_PRP4_LBSTK_FROM_12  0x68c
```



```
 317 #define MSR_PRP4_LBSTK_FROM_13  0x68d
 318 #define MSR_PRP4_LBSTK_FROM_14  0x68e
 319 #define MSR_PRP4_LBSTK_FROM_15  0x68f
 320 #define MSR_PRP4_LBSTK_TO_0     0x6c0
 321 #define MSR_PRP4_LBSTK_TO_1     0x6c1
 322 #define MSR_PRP4_LBSTK_TO_2     0x6c2
 323 #define MSR_PRP4_LBSTK_TO_3     0x6c3
 324 #define MSR_PRP4_LBSTK_TO_4     0x6c4
 325 #define MSR_PRP4_LBSTK_TO_5     0x6c5
 326 #define MSR_PRP4_LBSTK_TO_6     0x6c6
 327 #define MSR_PRP4_LBSTK_TO_7     0x6c7
 328 #define MSR_PRP4_LBSTK_TO_8     0x6c8
 329 #define MSR_PRP4_LBSTK_TO_9     0x6c9
 330 #define MSR_PRP4_LBSTK_TO_10    0x6ca
 331 #define MSR_PRP4_LBSTK_TO_11    0x6cb
 332 #define MSR_PRP4_LBSTK_TO_12    0x6cc
 333 #define MSR_PRP4_LBSTK_TO_13    0x6cd
 334 #define MSR_PRP4_LBSTK_TO_14    0x6ce
 335 #define MSR_PRP4_LBSTK_TO_15    0x6cf

 337 #define MCI_CTL_VALUE           0xffffffff

 339 #define MTRR_TYPE_UC            0
 340 #define MTRR_TYPE_WC            1
 341 #define MTRR_TYPE_WT            4
 342 #define MTRR_TYPE_WP            5
 343 #define MTRR_TYPE_WB            6
 344 #define MTRR_TYPE_UC_           7

 346 /*
 347  * For Solaris we set up the page attritubute table in the following way:
 348  * PAT0 Write-Back
 349  * PAT1 Write-Through
 350  * PAT2 Unchacheable-
 351  * PAT3 Uncacheable
 352  * PAT4 Write-Back
 353  * PAT5 Write-Through
 354  * PAT6 Write-Combine
 355  * PAT7 Uncacheable
 356  * The only difference from h/w default is entry 6.
 357  */
 358 #define PAT_DEFAULT_ATTRIBUTE                   \
 359         ((uint64_t)MTRR_TYPE_WB |               \
 360         ((uint64_t)MTRR_TYPE_WT << 8) |         \
 361         ((uint64_t)MTRR_TYPE_UC_ << 16) |       \
 362         ((uint64_t)MTRR_TYPE_UC << 24) |        \
 363         ((uint64_t)MTRR_TYPE_WB << 32) |        \
 364         ((uint64_t)MTRR_TYPE_WT << 40) |        \
 365         ((uint64_t)MTRR_TYPE_WC << 48) |        \
 366         ((uint64_t)MTRR_TYPE_UC << 56))

 368 #define X86FSET_LARGEPAGE       0
 369 #define X86FSET_TSC             1
 370 #define X86FSET_MSR             2
 371 #define X86FSET_MTRR            3
 372 #define X86FSET_PGE             4
 373 #define X86FSET_DE              5
 374 #define X86FSET_CMOV            6
 375 #define X86FSET_MMX             7
 376 #define X86FSET_MCA             8
 377 #define X86FSET_PAE             9
 378 #define X86FSET_CX8             10
 379 #define X86FSET_PAT             11
 380 #define X86FSET_SEP             12
 381 #define X86FSET_SSE             13
 382 #define X86FSET_SSE2            14
```

```
383 #define X86FSET_HTT             15
384 #define X86FSET_ASYSC           16
385 #define X86FSET_NX              17
386 #define X86FSET_SSE3            18
387 #define X86FSET_CX16            19
388 #define X86FSET_CMP             20
389 #define X86FSET_TSCP            21
390 #define X86FSET_MWAIT           22
391 #define X86FSET_SSE4A           23
392 #define X86FSET_CPUID           24
393 #define X86FSET_SSSE3           25
394 #define X86FSET_SSE4_1          26
395 #define X86FSET_SSE4_2          27
396 #define X86FSET_1GPG            28
397 #define X86FSET_CLFSH           29
398 #define X86FSET_64              30
399 #define X86FSET_AES             31
400 #define X86FSET_PCLMULQDQ       32
401 #define X86FSET_XSAVE           33
402 #define X86FSET_AVX             34
403 #define X86FSET_VMX             35
404 #define X86FSET_SVM             36
405 #define X86FSET_TOPOEXT         37
406 #define X86FSET_F16C            38
407 #define X86FSET_RDRAND          39
408 #define X86FSET_X2APIC          40
409 #define X86FSET_AVX2            41
410 #define X86FSET_BMI1            42
411 #define X86FSET_BMI2            43
412 #define X86FSET_FMA             44
413 #define X86FSET_SMEP            45
414 #define X86FSET_SMAP            46
415 #define X86FSET_ADX             47
416 #define X86FSET_RDSEED          48
417 #define X86FSET_MPX             49
418 #define X86FSET_AVX512F         50
419 #define X86FSET_AVX512DQ        51
420 #define X86FSET_AVX512PF        52
421 #define X86FSET_AVX512ER        53
422 #define X86FSET_AVX512CD        54
423 #define X86FSET_AVX512BW        55
424 #define X86FSET_AVX512VL        56
425 #define X86FSET_AVX512FMA       57
426 #define X86FSET_AVX512VBMI      58
427 #define X86FSET_AVX512VPOPCDQ   59
428 #define X86FSET_AVX512NNIW      60
429 #define X86FSET_AVX512FMAPS     61
430 #define X86FSET_XSAVEOPT        62
431 #define X86FSET_XSAVEC          63
432 #define X86FSET_XSAVES          64
433 #define X86FSET_SHA             65
434 #define X86FSET_UMIP            66
435 #define X86FSET_PKU             67
436 #define X86FSET_OSPKE           68
437 #define X86FSET_PCID            69
438 #define X86FSET_INVPCID         70

440 /*
441  * Intel Deep C-State invariant TSC in leaf 0x80000007.
442  */
443 #define CPUID_TSC_CSTATE_INVARIANCE     (0x100)

445 /*
446  * Intel Deep C-state always-running local APIC timer
447  */
448 #define CPUID_CSTATE_ARAT       (0x4)
```



```
450 /*
451  * Intel ENERGY_PERF_BIAS MSR indicated by feature bit CPUID.6.ECX[3].
452  */
453 #define CPUID_EPB_SUPPORT       (1 << 3)

455 /*
456  * Intel TSC deadline timer
457  */
458 #define CPUID_DEADLINE_TSC      (1 << 24)

460 /*
461  * x86_type is a legacy concept; this is supplanted
462  * for most purposes by x86_featureset; modern CPUs
463  * should be X86_TYPE_OTHER
464  */
465 #define X86_TYPE_OTHER          0
466 #define X86_TYPE_486            1
467 #define X86_TYPE_P5             2
468 #define X86_TYPE_P6             3
469 #define X86_TYPE_CYRIX_486      4
470 #define X86_TYPE_CYRIX_6x86L    5
471 #define X86_TYPE_CYRIX_6x86     6
472 #define X86_TYPE_CYRIX_GXm      7
473 #define X86_TYPE_CYRIX_6x86MX   8
474 #define X86_TYPE_CYRIX_MediaGX  9
475 #define X86_TYPE_CYRIX_MII      10
476 #define X86_TYPE_VIA_CYRIX_III  11
477 #define X86_TYPE_P4             12

479 /*
480  * x86_vendor allows us to select between
481  * implementation features and helps guide
482  * the interpretation of the cpuid instruction.
483  */
484 #define X86_VENDOR_Intel        0
485 #define X86_VENDORSTR_Intel     "GenuineIntel"

487 #define X86_VENDOR_IntelClone   1

489 #define X86_VENDOR_AMD          2
490 #define X86_VENDORSTR_AMD       "AuthenticAMD"

492 #define X86_VENDOR_Cyrix        3
493 #define X86_VENDORSTR_CYRIX     "CyrixInstead"

495 #define X86_VENDOR_UMC          4
496 #define X86_VENDORSTR_UMC       "UMC UMC UMC "

498 #define X86_VENDOR_NexGen       5
499 #define X86_VENDORSTR_NexGen    "NexGenDriven"

501 #define X86_VENDOR_Centaur      6
502 #define X86_VENDORSTR_Centaur   "CentaurHauls"

504 #define X86_VENDOR_Rise         7
505 #define X86_VENDORSTR_Rise      "RiseRiseRise"

507 #define X86_VENDOR_SiS          8
508 #define X86_VENDORSTR_SiS       "SiS SiS SiS "

510 #define X86_VENDOR_TM           9
511 #define X86_VENDORSTR_TM        "GenuineTMx86"

513 #define X86_VENDOR_NSC          10
514 #define X86_VENDORSTR_NSC       "Geode by NSC"
```

```
 516 /*
 517  * Vendor string max len + \0
 518  */
 519 #define X86_VENDOR_STRLEN       13

 521 /*
 522  * Some vendor/family/model/stepping ranges are commonly grouped under
 523  * a single identifying banner by the vendor.  The following encode
 524  * that "revision" in a uint32_t with the 8 most significant bits
 525  * identifying the vendor with X86_VENDOR_*, the next 8 identifying the
 526  * family, and the remaining 16 typically forming a bitmask of revisions
 527  * within that family with more significant bits indicating "later" revisions.
 528  */

 530 #define _X86_CHIPREV_VENDOR_MASK        0xff000000u
 531 #define _X86_CHIPREV_VENDOR_SHIFT       24
 532 #define _X86_CHIPREV_FAMILY_MASK        0x00ff0000u
 533 #define _X86_CHIPREV_FAMILY_SHIFT       16
 534 #define _X86_CHIPREV_REV_MASK           0x0000ffffu

 536 #define _X86_CHIPREV_VENDOR(x) \
 537         (((x) & _X86_CHIPREV_VENDOR_MASK) >> _X86_CHIPREV_VENDOR_SHIFT)
 538 #define _X86_CHIPREV_FAMILY(x) \
 539         (((x) & _X86_CHIPREV_FAMILY_MASK) >> _X86_CHIPREV_FAMILY_SHIFT)
 540 #define _X86_CHIPREV_REV(x) \
 541         ((x) & _X86_CHIPREV_REV_MASK)

 543 /* True if x matches in vendor and family and if x matches the given rev mask */
 544 #define X86_CHIPREV_MATCH(x, mask) \
 545         (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(mask) && \
 546         _X86_CHIPREV_FAMILY(x) == _X86_CHIPREV_FAMILY(mask) && \
 547         ((_X86_CHIPREV_REV(x) & _X86_CHIPREV_REV(mask)) != 0))

 549 /* True if x matches in vendor and family, and rev is at least minx */
 550 #define X86_CHIPREV_ATLEAST(x, minx) \
 551         (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(minx) && \
 552         _X86_CHIPREV_FAMILY(x) == _X86_CHIPREV_FAMILY(minx) && \
 553         _X86_CHIPREV_REV(x) >= _X86_CHIPREV_REV(minx))

 555 #define _X86_CHIPREV_MKREV(vendor, family, rev) \
 556         ((uint32_t)(vendor) << _X86_CHIPREV_VENDOR_SHIFT | \
 557         (family) << _X86_CHIPREV_FAMILY_SHIFT | (rev))

 559 /* True if x matches in vendor, and family is at least minx */
 560 #define X86_CHIPFAM_ATLEAST(x, minx) \
 561         (_X86_CHIPREV_VENDOR(x) == _X86_CHIPREV_VENDOR(minx) && \
 562         _X86_CHIPREV_FAMILY(x) >= _X86_CHIPREV_FAMILY(minx))

 564 /* Revision default */
 565 #define X86_CHIPREV_UNKNOWN     0x0

 567 /*
 568  * Definitions for AMD Family 0xf. Minor revisions C0 and CG are
 569  * sufficiently different that we will distinguish them; in all other
 570  * case we will identify the major revision.
 571  */
 572 #define X86_CHIPREV_AMD_F_REV_B _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0001)
 573 #define X86_CHIPREV_AMD_F_REV_C0 _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0002)
 574 #define X86_CHIPREV_AMD_F_REV_CG _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0004)
 575 #define X86_CHIPREV_AMD_F_REV_D _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0008)
 576 #define X86_CHIPREV_AMD_F_REV_E _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0010)
 577 #define X86_CHIPREV_AMD_F_REV_F _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0020)
 578 #define X86_CHIPREV_AMD_F_REV_G _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0xf, 0x0040)

 580 /*
```



```
 581  * Definitions for AMD Family 0x10.  Rev A was Engineering Samples only.
 582  */
 583 #define X86_CHIPREV_AMD_10_REV_A \
 584         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0001)
 585 #define X86_CHIPREV_AMD_10_REV_B \
 586         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0002)
 587 #define X86_CHIPREV_AMD_10_REV_C2 \
 588         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0004)
 589 #define X86_CHIPREV_AMD_10_REV_C3 \
 590         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0008)
 591 #define X86_CHIPREV_AMD_10_REV_D0 \
 592         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0010)
 593 #define X86_CHIPREV_AMD_10_REV_D1 \
 594         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0020)
 595 #define X86_CHIPREV_AMD_10_REV_E \
 596         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x10, 0x0040)

 598 /*
 599  * Definitions for AMD Family 0x11.
 600  */
 601 #define X86_CHIPREV_AMD_11_REV_B \
 602         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x11, 0x0002)

 604 /*
 605  * Definitions for AMD Family 0x12.
 606  */
 607 #define X86_CHIPREV_AMD_12_REV_B \
 608         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x12, 0x0002)

 610 /*
 611  * Definitions for AMD Family 0x14.
 612  */
 613 #define X86_CHIPREV_AMD_14_REV_B \
 614         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x14, 0x0002)
 615 #define X86_CHIPREV_AMD_14_REV_C \
 616         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x14, 0x0004)

 618 /*
 619  * Definitions for AMD Family 0x15
 620  */
 621 #define X86_CHIPREV_AMD_15OR_REV_B2 \
 622         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0001)

 624 #define X86_CHIPREV_AMD_15TN_REV_A1 \
 625         _X86_CHIPREV_MKREV(X86_VENDOR_AMD, 0x15, 0x0002)

 627 /*
 628  * Various socket/package types, extended as the need to distinguish
 629  * a new type arises.  The top 8 byte identfies the vendor and the
 630  * remaining 24 bits describe 24 socket types.
 631  */

 633 #define _X86_SOCKET_VENDOR_SHIFT        24
 634 #define _X86_SOCKET_VENDOR(x)   ((x) >> _X86_SOCKET_VENDOR_SHIFT)
 635 #define _X86_SOCKET_TYPE_MASK   0x00ffffff
 636 #define _X86_SOCKET_TYPE(x)             ((x) & _X86_SOCKET_TYPE_MASK)

 638 #define _X86_SOCKET_MKVAL(vendor, bitval) \
 639         ((uint32_t)(vendor) << _X86_SOCKET_VENDOR_SHIFT | (bitval))

 641 #define X86_SOCKET_MATCH(s, mask) \
 642         (_X86_SOCKET_VENDOR(s) == _X86_SOCKET_VENDOR(mask) && \
 643         (_X86_SOCKET_TYPE(s) & _X86_SOCKET_TYPE(mask)) != 0)

 645 #define X86_SOCKET_UNKNOWN 0x0
 646         /*
```

```
 647          * AMD socket types
 648          */
 649 #define X86_SOCKET_754          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000001)
 650 #define X86_SOCKET_939          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000002)
 651 #define X86_SOCKET_940          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000004)
 652 #define X86_SOCKET_S1g1         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000008)
 653 #define X86_SOCKET_AM2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000010)
 654 #define X86_SOCKET_F1207        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000020)
 655 #define X86_SOCKET_S1g2         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000040)
 656 #define X86_SOCKET_S1g3         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000080)
 657 #define X86_SOCKET_AM           _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000100)
 658 #define X86_SOCKET_AM2R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000200)
 659 #define X86_SOCKET_AM3          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000400)
 660 #define X86_SOCKET_G34          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x000800)
 661 #define X86_SOCKET_ASB2         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x001000)
 662 #define X86_SOCKET_C32          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x002000)
 663 #define X86_SOCKET_S1g4         _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x004000)
 664 #define X86_SOCKET_FT1          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x008000)
 665 #define X86_SOCKET_FM1          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x010000)
 666 #define X86_SOCKET_FS1          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x020000)
 667 #define X86_SOCKET_AM3R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x040000)
 668 #define X86_SOCKET_FP2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x080000)
 669 #define X86_SOCKET_FS1R2        _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x100000)
 670 #define X86_SOCKET_FM2          _X86_SOCKET_MKVAL(X86_VENDOR_AMD, 0x200000)

 672 /*
 673  * xgetbv/xsetbv support
 674  * See section 13.3 in vol. 1 of the Intel devlopers manual.
 675  */

 677 #define XFEATURE_ENABLED_MASK   0x0
 678 /*
 679  * XFEATURE_ENABLED_MASK values (eax)
 680  * See setup_xfem().
 681  */
 682 #define XFEATURE_LEGACY_FP      0x1
 683 #define XFEATURE_SSE            0x2
 684 #define XFEATURE_AVX            0x4
 685 #define XFEATURE_MPX            0x18    /* 2 bits, both 0 or 1 */
 686 #define XFEATURE_AVX512         0xe0    /* 3 bits, all 0 or 1 */
 687         /* bit 8 unused */
 688 #define XFEATURE_PKRU           0x200
 689 #define XFEATURE_FP_ALL \
 690         (XFEATURE_LEGACY_FP | XFEATURE_SSE | XFEATURE_AVX | XFEATURE_MPX | \
 691         XFEATURE_AVX512 | XFEATURE_PKRU)

 693 #if !defined(_ASM)

 695 #if defined(_KERNEL) || defined(_KMEMUSER)

 697 #define NUM_X86_FEATURES        71
 685 #define NUM_X86_FEATURES        69
 698 extern uchar_t x86_featureset[];

 700 extern void free_x86_featureset(void *featureset);
 701 extern boolean_t is_x86_feature(void *featureset, uint_t feature);
 702 extern void add_x86_feature(void *featureset, uint_t feature);
 703 extern void remove_x86_feature(void *featureset, uint_t feature);
 704 extern boolean_t compare_x86_featureset(void *setA, void *setB);
 705 extern void print_x86_featureset(void *featureset);


 708 extern uint_t x86_type;
 709 extern uint_t x86_vendor;
 710 extern uint_t x86_clflush_size;
```



```
 712 extern uint_t pentiumpro_bug4046376;

 714 extern const char CyrixInstead[];

 716 #endif

 718 #if defined(_KERNEL)

 720 /*
 721  * This structure is used to pass arguments and get return values back
 722  * from the CPUID instruction in __cpuid_insn() routine.
 723  */
 724 struct cpuid_regs {
 725         uint32_t        cp_eax;
 726         uint32_t        cp_ebx;
 727         uint32_t        cp_ecx;
 728         uint32_t        cp_edx;
 729 };

 731 extern int x86_use_pcid;
 732 extern int x86_use_invpcid;

 734 /*
 735  * Utility functions to get/set extended control registers (XCR)
 736  * Initial use is to get/set the contents of the XFEATURE_ENABLED_MASK.
 737  */
 738 extern uint64_t get_xcr(uint_t);
 739 extern void set_xcr(uint_t, uint64_t);

 741 extern uint64_t rdmsr(uint_t);
 742 extern void wrmsr(uint_t, const uint64_t);
 743 extern uint64_t xrdmsr(uint_t);
 744 extern void xwrmsr(uint_t, const uint64_t);
 745 extern int checked_rdmsr(uint_t, uint64_t *);
 746 extern int checked_wrmsr(uint_t, uint64_t);

 748 extern void invalidate_cache(void);
 749 extern ulong_t getcr4(void);
 750 extern void setcr4(ulong_t);

 752 extern void mtrr_sync(void);

 754 extern void cpu_fast_syscall_enable(void *);
 755 extern void cpu_fast_syscall_disable(void *);

 757 struct cpu;

 759 extern int cpuid_checkpass(struct cpu *, int);
 760 extern uint32_t cpuid_insn(struct cpu *, struct cpuid_regs *);
 761 extern uint32_t __cpuid_insn(struct cpuid_regs *);
 762 extern int cpuid_getbrandstr(struct cpu *, char *, size_t);
 763 extern int cpuid_getidstr(struct cpu *, char *, size_t);
 764 extern const char *cpuid_getvendorstr(struct cpu *);
 765 extern uint_t cpuid_getvendor(struct cpu *);
 766 extern uint_t cpuid_getfamily(struct cpu *);
 767 extern uint_t cpuid_getmodel(struct cpu *);
 768 extern uint_t cpuid_getstep(struct cpu *);
 769 extern uint_t cpuid_getsig(struct cpu *);
 770 extern uint_t cpuid_get_ncpu_per_chip(struct cpu *);
 771 extern uint_t cpuid_get_ncore_per_chip(struct cpu *);
 772 extern uint_t cpuid_get_ncpu_sharing_last_cache(struct cpu *);
 773 extern id_t cpuid_get_last_lvl_cacheid(struct cpu *);
 774 extern int cpuid_get_chipid(struct cpu *);
 775 extern id_t cpuid_get_coreid(struct cpu *);
 776 extern int cpuid_get_pkgcoreid(struct cpu *);
 777 extern int cpuid_get_clogid(struct cpu *);
```

```
 778 extern int cpuid_get_cacheid(struct cpu *);
 779 extern uint32_t cpuid_get_apicid(struct cpu *);
 780 extern uint_t cpuid_get_procnodeid(struct cpu *cpu);
 781 extern uint_t cpuid_get_procnodes_per_pkg(struct cpu *cpu);
 782 extern uint_t cpuid_get_compunitid(struct cpu *cpu);
 783 extern uint_t cpuid_get_cores_per_compunit(struct cpu *cpu);
 784 extern size_t cpuid_get_xsave_size();
 785 extern boolean_t cpuid_need_fp_excp_handling();
 786 extern int cpuid_is_cmt(struct cpu *);
 787 extern int cpuid_syscall32_insn(struct cpu *);
 788 extern int getl2cacheinfo(struct cpu *, int *, int *, int *);

 790 extern uint32_t cpuid_getchiprev(struct cpu *);
 791 extern const char *cpuid_getchiprevstr(struct cpu *);
 792 extern uint32_t cpuid_getsockettype(struct cpu *);
 793 extern const char *cpuid_getsocketstr(struct cpu *);

 795 extern int cpuid_have_cr8access(struct cpu *);

 797 extern int cpuid_opteron_erratum(struct cpu *, uint_t);

 799 struct cpuid_info;

 801 extern void setx86isalist(void);
 802 extern void cpuid_alloc_space(struct cpu *);
 803 extern void cpuid_free_space(struct cpu *);
 804 extern void cpuid_pass1(struct cpu *, uchar_t *);
 805 extern void cpuid_pass2(struct cpu *);
 806 extern void cpuid_pass3(struct cpu *);
 807 extern void cpuid_pass4(struct cpu *, uint_t *);
 808 extern void cpuid_set_cpu_properties(void *, processorid_t,
 809     struct cpuid_info *);

 811 extern void cpuid_get_addrsize(struct cpu *, uint_t *, uint_t *);
 812 extern uint_t cpuid_get_dtlb_nent(struct cpu *, size_t);

 814 #if !defined(__xpv)
 815 extern uint32_t *cpuid_mwait_alloc(struct cpu *);
 816 extern void cpuid_mwait_free(struct cpu *);
 817 extern int cpuid_deep_cstates_supported(void);
 818 extern int cpuid_arat_supported(void);
 819 extern int cpuid_iepb_supported(struct cpu *);
 820 extern int cpuid_deadline_tsc_supported(void);
 821 extern void vmware_port(int, uint32_t *);
 822 #endif

 824 struct cpu_ucode_info;

 826 extern void ucode_alloc_space(struct cpu *);
 827 extern void ucode_free_space(struct cpu *);
 828 extern void ucode_check(struct cpu *);
 829 extern void ucode_cleanup();

 831 #if !defined(__xpv)
 832 extern  char _tsc_mfence_start;
 833 extern  char _tsc_mfence_end;
 834 extern  char _tscp_start;
 835 extern  char _tscp_end;
 836 extern  char _no_rdtsc_start;
 837 extern  char _no_rdtsc_end;
 838 extern  char _tsc_lfence_start;
 839 extern  char _tsc_lfence_end;
 840 #endif

 842 #if !defined(__xpv)
 843 extern  char bcopy_patch_start;
```



```
 844 extern  char bcopy_patch_end;
 845 extern  char bcopy_ck_size;
 846 #endif

 848 extern void post_startup_cpu_fixups(void);

 850 extern uint_t workaround_errata(struct cpu *);

 852 #if defined(OPTERON_ERRATUM_93)
 853 extern int opteron_erratum_93;
 854 #endif

 856 #if defined(OPTERON_ERRATUM_91)
 857 extern int opteron_erratum_91;
 858 #endif

 860 #if defined(OPTERON_ERRATUM_100)
 861 extern int opteron_erratum_100;
 862 #endif

 864 #if defined(OPTERON_ERRATUM_121)
 865 extern int opteron_erratum_121;
 866 #endif

 868 #if defined(OPTERON_WORKAROUND_6323525)
 869 extern int opteron_workaround_6323525;
 870 extern void patch_workaround_6323525(void);
 871 #endif

 873 #if !defined(__xpv)
 874 extern void determine_platform(void);
 875 #endif
 876 extern int get_hwenv(void);
 877 extern int is_controldom(void);

 879 extern void enable_pcid(void);

 881 extern void xsave_setup_msr(struct cpu *);

 883 /*
 884  * Hypervisor signatures
 885  */
 886 #define HVSIG_XEN_HVM   "XenVMMXenVMM"
 887 #define HVSIG_VMWARE    "VMwareVMware"
 888 #define HVSIG_KVM       "KVMKVMKVM"
 889 #define HVSIG_MICROSOFT "Microsoft Hv"

 891 /*
 892  * Defined hardware environments
 893  */
 894 #define HW_NATIVE       (1 << 0)        /* Running on bare metal */
 895 #define HW_XEN_PV       (1 << 1)        /* Running on Xen PVM */

 897 #define HW_XEN_HVM      (1 << 2)        /* Running on Xen HVM */
 898 #define HW_VMWARE       (1 << 3)        /* Running on VMware hypervisor */
 899 #define HW_KVM          (1 << 4)        /* Running on KVM hypervisor */
 900 #define HW_MICROSOFT    (1 << 5)        /* Running on Microsoft hypervisor */

 902 #define HW_VIRTUAL      (HW_XEN_HVM | HW_VMWARE | HW_KVM | HW_MICROSOFT)

 904 #endif  /* _KERNEL */

 906 #endif  /* !_ASM */

 908 /*
 909  * VMware hypervisor related defines
```

```
 910  */
 911 #define VMWARE_HVMAGIC          0x564d5868
 912 #define VMWARE_HVPORT           0x5658
 913 #define VMWARE_HVCMD_GETVERSION 0x0a
 914 #define VMWARE_HVCMD_GETTSCFREQ 0x2d

 916 #ifdef  __cplusplus
 917 }
```

_____***unchanged_portion_omitted_***