```
**********************************************************
   47168 Sat Mar 30 14:54:00 2019
new/usr/src/uts/i86pc/io/pcplusmp/apic_common.c
10597 would like a way to set NMI behavior at boot
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Andy Fiddaman <andy@omniosce.org>
**********************************************************
```

```
/*
 * 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) 2010, Oracle and/or its affiliates. All rights reserved.
 */
/*
 * Copyright 2019, Joyent, Inc.
 * Copyright 2018 Joyent, Inc.
 * Copyright (c) 2016, 2017 by Delphix. All rights reserved.
 */

/*
 * PSMI 1.1 extensions are supported only in 2.6 and later versions.
 * PSMI 1.2 extensions are supported only in 2.7 and later versions.
 * PSMI 1.3 and 1.4 extensions are supported in Solaris 10.
 * PSMI 1.5 extensions are supported in Solaris Nevada.
 * PSMI 1.6 extensions are supported in Solaris Nevada.
 * PSMI 1.7 extensions are supported in Solaris Nevada.
 */
#define PSMI_1_7

#include <sys/processor.h>
#include <sys/time.h>
#include <sys/psm.h>
#include <sys/smp_impldefs.h>
#include <sys/cram.h>
#include <sys/acpi/acpi.h>
#include <sys/acpica.h>
#include <sys/psm_common.h>
#include <sys/apic.h>
#include <sys/pit.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/ddi_impldefs.h>
#include <sys/pci.h>
#include <sys/promif.h>
#include <sys/x86_archext.h>
#include <sys/cpc_impl.h>
#include <sys/uadmin.h>
#include <sys/panic.h>
```



```
#include <sys/debug.h>
#include <sys/archsystm.h>
#include <sys/trap.h>
#include <sys/machsystm.h>
#include <sys/sysmacros.h>
#include <sys/cpuvar.h>
#include <sys/rm_platter.h>
#include <sys/privregs.h>
#include <sys/note.h>
#include <sys/pci_intr_lib.h>
#include <sys/spl.h>
#include <sys/clock.h>
#include <sys/dditypes.h>
#include <sys/sunddi.h>
#include <sys/x_call.h>
#include <sys/reboot.h>
#include <sys/hpet.h>
#include <sys/apic_common.h>
#include <sys/apic_timer.h>

static void     apic_record_ioapic_rdt(void *intrmap_private,
                    ioapic_rdt_t *irdt);
static void     apic_record_msi(void *intrmap_private, msi_regs_t *mregs);

/*
 * Common routines between pcplusmp & apix (taken from apic.c).
 */

int     apic_clkinit(int);
hrtime_t apic_gethrtime(void);
void    apic_send_ipi(int, int);
void    apic_set_idlecpu(processorid_t);
void    apic_unset_idlecpu(processorid_t);
void    apic_shutdown(int, int);
void    apic_preshutdown(int, int);
processorid_t   apic_get_next_processorid(processorid_t);

hrtime_t apic_gettime();

enum apic_ioapic_method_type apix_mul_ioapic_method = APIC_MUL_IOAPIC_PCPLUSMP;

/* Now the ones for Dynamic Interrupt distribution */
int     apic_enable_dynamic_migration = 0;

/* maximum loop count when sending Start IPIs. */
int apic_sipi_max_loop_count = 0x1000;

/*
 * These variables are frequently accessed in apic_intr_enter(),
 * apic_intr_exit and apic_setspl, so group them together
 */
volatile uint32_t *apicadr =  NULL;     /* virtual addr of local APIC   */
int apic_setspl_delay = 1;              /* apic_setspl - delay enable   */
int apic_clkvect;

/* vector at which error interrupts come in */
int apic_errvect;
int apic_enable_error_intr = 1;
int apic_error_display_delay = 100;

/* vector at which performance counter overflow interrupts come in */
int apic_cpcovf_vect;
int apic_enable_cpcovf_intr = 1;

/* vector at which CMCI interrupts come in */
int apic_cmci_vect;
```

```
125 extern void cmi_cmci_trap(void);

127 lock_t apic_mode_switch_lock;

129 int apic_pir_vect;

131 /*
132  * Patchable global variables.
133  */
134 int     apic_forceload = 0;

136 int     apic_coarse_hrtime = 1;         /* 0 - use accurate slow gethrtime() */

138 int     apic_flat_model = 0;            /* 0 - clustered. 1 - flat */
139 int     apic_panic_on_nmi = 0;
140 int     apic_panic_on_apic_error = 0;

142 int     apic_verbose = 0;       /* 0x1ff */

144 #ifdef DEBUG
145 int     apic_debug = 0;
146 int     apic_restrict_vector = 0;

148 int     apic_debug_msgbuf[APIC_DEBUG_MSGBUFSIZE];
149 int     apic_debug_msgbufindex = 0;

151 #endif /* DEBUG */

153 uint_t apic_nticks = 0;
154 uint_t apic_skipped_redistribute = 0;

156 uint_t last_count_read = 0;
157 lock_t  apic_gethrtime_lock;
158 volatile int    apic_hrtime_stamp = 0;
159 volatile hrtime_t apic_nsec_since_boot = 0;

161 static  hrtime_t        apic_last_hrtime = 0;
162 int             apic_hrtime_error = 0;
163 int             apic_remote_hrterr = 0;
164 int             apic_num_nmis = 0;
165 int             apic_apic_error = 0;
166 int             apic_num_apic_errors = 0;
167 int             apic_num_cksum_errors = 0;

169 int     apic_error = 0;

171 static  int     apic_cmos_ssb_set = 0;

173 /* use to make sure only one cpu handles the nmi */
174 lock_t  apic_nmi_lock;
175 /* use to make sure only one cpu handles the error interrupt */
176 lock_t  apic_error_lock;

178 static  struct {
179         uchar_t cntl;
180         uchar_t data;
181 } aspen_bmc[] = {
```

*_____unchanged_portion_omitted_*

```
806 /* apic NMI handler */
807 /*ARGSUSED*/
808 void
809 apic_nmi_intr(caddr_t arg, struct regs *rp)
810 {
811         nmi_action_t action = nmi_action;
```



```
813         if (apic_shutdown_processors) {
814                 apic_disable_local_apic();
815                 return;
816         }

818         apic_error |= APIC_ERR_NMI;

820         if (!lock_try(&apic_nmi_lock))
821                 return;
822         apic_num_nmis++;

822         if (apic_kmdb_on_nmi && psm_debugger()) {
823                 debug_enter("NMI received: entering kmdb\n");
824         } else if (apic_panic_on_nmi) {
825                 /* Keep panic from entering kmdb. */
826                 nopanicdebug = 1;
827                 panic("NMI received\n");
828         } else {
824         /*
825          * "nmi_action" always over-rides the older way of doing this, unless we
826          * can't actually drop into kmdb when requested.
827          */
828         if (action == NMI_ACTION_KMDB && !psm_debugger())
829                 action = NMI_ACTION_UNSET;

831         if (action == NMI_ACTION_UNSET) {
832                 if (apic_kmdb_on_nmi && psm_debugger())
833                         action = NMI_ACTION_KMDB;
834                 else if (apic_panic_on_nmi)
835                         action = NMI_ACTION_PANIC;
836                 else
837                         action = NMI_ACTION_IGNORE;
838         }

840         switch (action) {
841         case NMI_ACTION_IGNORE:
842                 /*
843                  * prom_printf is the best shot we have of something which is
844                  * problem free from high level/NMI type of interrupts
845                  */
846                 prom_printf("NMI received\n");
847                 break;

849         case NMI_ACTION_PANIC:
850                 /* Keep panic from entering kmdb. */
851                 nopanicdebug = 1;
852                 panic("NMI received\n");
853                 break;

855         case NMI_ACTION_KMDB:
856         default:
857                 debug_enter("NMI received: entering kmdb\n");
858                 break;
859         }

861         lock_clear(&apic_nmi_lock);
862 }
```

*_____unchanged_portion_omitted_*

```
*********************************************************
   70392 Sat Mar 30 14:54:01 2019
new/usr/src/uts/i86pc/os/fakebop.c
10597 would like a way to set NMI behavior at boot
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Andy Fiddaman <andy@omniosce.org>
*********************************************************
______unchanged_portion_omitted_

2918 #define BP_MAX_STRLEN   32

2918 /*
2919  * Get an integer value for given boot property
2921  * Get value for given boot property
2920  */
2921 int
2922 bootprop_getval(const char *prop_name, u_longlong_t *prop_value)
2923 {
2924         int             boot_prop_len;
2925         char            str[BP_MAX_STRLEN];
2926         u_longlong_t    value;

2928         boot_prop_len = BOP_GETPROPLEN(bootops, prop_name);
2929         if (boot_prop_len < 0 || boot_prop_len >= sizeof (str) ||
2931         if (boot_prop_len < 0 || boot_prop_len > sizeof (str) ||
2930             BOP_GETPROP(bootops, prop_name, str) < 0 ||
2931             kobj_getvalue(str, &value) == -1)
2932                 return (-1);

2934         if (prop_value)
2935                 *prop_value = value;

2937         return (0);
2938 }

2940 int
2941 bootprop_getstr(const char *prop_name, char *buf, size_t buflen)
2942 {
2943         int boot_prop_len = BOP_GETPROPLEN(bootops, prop_name);

2945         if (boot_prop_len < 0 || boot_prop_len >= buflen ||
2946             BOP_GETPROP(bootops, prop_name, buf) < 0)
2947                 return (-1);

2949         return (0);
2950 }
______unchanged_portion_omitted_
```

```
**********************************************************
   15349 Sat Mar 30 14:54:02 2019
new/usr/src/uts/i86pc/os/mlsetup.c
10597 would like a way to set NMI behavior at boot
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Andy Fiddaman <andy@omniosce.org>
**********************************************************
```

```
/*
 * 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 2019, Joyent, Inc.
 * Copyright 2018 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/apic.h>
#include <sys/apic_common.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;

nmi_action_t nmi_action = NMI_ACTION_UNSET;

/*
 * 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;
        char prop_str[BP_MAX_STRLEN];
        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)
```

```
125         /*
126          * Point at the hypervisor's virtual cpu structure
127          */
128         cpu[0]->cpu_m.mcpu_vcpu_info = &HYPERVISOR_shared_info->vcpu_info[0];
129 #endif

131         /*
132          * check if we've got special bits to clear or set
133          * when checking cpu features
134          */

136         if (bootprop_getval("cpuid_feature_ecx_include", &prop_value) != 0)
137                 cpuid_feature_ecx_include = 0;
138         else
139                 cpuid_feature_ecx_include = (uint32_t)prop_value;

141         if (bootprop_getval("cpuid_feature_ecx_exclude", &prop_value) != 0)
142                 cpuid_feature_ecx_exclude = 0;
143         else
144                 cpuid_feature_ecx_exclude = (uint32_t)prop_value;

146         if (bootprop_getval("cpuid_feature_edx_include", &prop_value) != 0)
147                 cpuid_feature_edx_include = 0;
148         else
149                 cpuid_feature_edx_include = (uint32_t)prop_value;

151         if (bootprop_getval("cpuid_feature_edx_exclude", &prop_value) != 0)
152                 cpuid_feature_edx_exclude = 0;
153         else
154                 cpuid_feature_edx_exclude = (uint32_t)prop_value;

156 #if !defined(__xpv)
157         if (bootprop_getstr("nmi", prop_str, sizeof (prop_str)) == 0) {
158                 if (strcmp(prop_str, "ignore") == 0) {
159                         nmi_action = NMI_ACTION_IGNORE;
160                 } else if (strcmp(prop_str, "panic") == 0) {
161                         nmi_action = NMI_ACTION_PANIC;
162                 } else if (strcmp(prop_str, "kmdb") == 0) {
163                         nmi_action = NMI_ACTION_KMDB;
164                 } else {
165                         prom_printf("unix: ignoring unknown nmi=%s\n",
166                             prop_str);
167                 }
168         }

170         /*
171          * Check to see if KPTI has been explicitly enabled or disabled.
172          * We have to check this before init_desctbls().
173          */
174         if (bootprop_getval("kpti", &prop_value) == 0) {
175                 kpti_enable = (uint64_t)(prop_value == 1);
176                 prom_printf("unix: forcing kpti to %s due to boot argument\n",
177                     (kpti_enable == 1) ? "ON" : "OFF");
178         } else {
179                 kpti_enable = 1;
180         }

182         if (bootprop_getval("pcid", &prop_value) == 0 && prop_value == 0) {
183                 prom_printf("unix: forcing pcid to OFF due to boot argument\n");
184                 x86_use_pcid = 0;
185         } else if (kpti_enable != 1) {
186                 x86_use_pcid = 0;
187         }
188 #endif

190         /*
```



```
191          * Initialize idt0, gdt0, ldt0_default, ktss0 and dftss.
192          */
193         init_desctbls();

195         /*
196          * lgrp_init() and possibly cpuid_pass1() need PCI config
197          * space access
198          */
199 #if defined(__xpv)
200         if (DOMAIN_IS_INITDOMAIN(xen_info))
201                 pci_cfgspace_init();
202 #else
203         pci_cfgspace_init();
204         /*
205          * Initialize the platform type from CPU 0 to ensure that
206          * determine_platform() is only ever called once.
207          */
208         determine_platform();
209 #endif

211         /*
212          * The first lightweight pass (pass0) through the cpuid data
213          * was done in locore before mlsetup was called.  Do the next
214          * pass in C code.
215          *
216          * The x86_featureset is initialized here based on the capabilities
217          * of the boot CPU.  Note that if we choose to support CPUs that have
218          * different feature sets (at which point we would almost certainly
219          * want to set the feature bits to correspond to the feature
220          * minimum) this value may be altered.
221          */
222         cpuid_pass1(cpu[0], x86_featureset);

224 #if !defined(__xpv)
225         if ((get_hwenv() & HW_XEN_HVM) != 0)
226                 xen_hvm_init();

228         /*
229          * Before we do anything with the TSCs, we need to work around
230          * Intel erratum BT81.  On some CPUs, warm reset does not
231          * clear the TSC.  If we are on such a CPU, we will clear TSC ourselves
232          * here.  Other CPUs will clear it when we boot them later, and the
233          * resulting skew will be handled by tsc_sync_master()/_slave();
234          * note that such skew already exists and has to be handled anyway.
235          *
236          * We do this only on metal.  This same problem can occur with a
237          * hypervisor that does not happen to virtualise a TSC that starts from
238          * zero, regardless of CPU type; however, we do not expect hypervisors
239          * that do not virtualise TSC that way to handle writes to TSC
240          * correctly, either.
241          */
242         if (get_hwenv() == HW_NATIVE &&
243             cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
244             cpuid_getfamily(CPU) == 6 &&
245             (cpuid_getmodel(CPU) == 0x2d || cpuid_getmodel(CPU) == 0x3e) &&
246             is_x86_feature(x86_featureset, X86FSET_TSC)) {
247                 (void) wrmsr(REG_TSC, 0UL);
248         }

250         /*
251          * Patch the tsc_read routine with appropriate set of instructions,
252          * depending on the processor family and architecure, to read the
253          * time-stamp counter while ensuring no out-of-order execution.
254          * Patch it while the kernel text is still writable.
255          *
256          * Note: tsc_read is not patched for intel processors whose family
```

```
257          * is >6 and for amd whose family >f (in case they don't support rdtscp
258          * instruction, unlikely). By default tsc_read will use cpuid for
259          * serialization in such cases. The following code needs to be
260          * revisited if intel processors of family >= f retains the
261          * instruction serialization nature of mfence instruction.
262          * Note: tsc_read is not patched for x86 processors which do
263          * not support "mfence". By default tsc_read will use cpuid for
264          * serialization in such cases.
265          *
266          * The Xen hypervisor does not correctly report whether rdtscp is
267          * supported or not, so we must assume that it is not.
268          */
269         if ((get_hwenv() & HW_XEN_HVM) == 0 &&
270             is_x86_feature(x86_featureset, X86FSET_TSCP))
271                 patch_tsc_read(TSC_TSCP);
272         else if (cpuid_getvendor(CPU) == X86_VENDOR_AMD &&
273             cpuid_getfamily(CPU) <= 0xf &&
274             is_x86_feature(x86_featureset, X86FSET_SSE2))
275                 patch_tsc_read(TSC_RDTSC_MFENCE);
276         else if (cpuid_getvendor(CPU) == X86_VENDOR_Intel &&
277             cpuid_getfamily(CPU) <= 6 &&
278             is_x86_feature(x86_featureset, X86FSET_SSE2))
279                 patch_tsc_read(TSC_RDTSC_LFENCE);

281 #endif  /* !__xpv */

283 #if defined(__i386) && !defined(__xpv)
284         /*
285          * Some i386 processors do not implement the rdtsc instruction,
286          * or at least they do not implement it correctly. Patch them to
287          * return 0.
288          */
289         if (!is_x86_feature(x86_featureset, X86FSET_TSC))
290                 patch_tsc_read(TSC_NONE);
291 #endif  /* __i386 && !__xpv */

293 #if defined(__amd64) && !defined(__xpv)
294         patch_memops(cpuid_getvendor(CPU));
295 #endif  /* __amd64 && !__xpv */

297 #if !defined(__xpv)
298         /* XXPV what, if anything, should be dorked with here under xen? */

300         /*
301          * While we're thinking about the TSC, let's set up %cr4 so that
302          * userland can issue rdtsc, and initialize the TSC_AUX value
303          * (the cpuid) for the rdtscp instruction on appropriately
304          * capable hardware.
305          */
306         if (is_x86_feature(x86_featureset, X86FSET_TSC))
307                 setcr4(getcr4() & ~CR4_TSD);

309         if (is_x86_feature(x86_featureset, X86FSET_TSCP))
310                 (void) wrmsr(MSR_AMD_TSCAUX, 0);

312         /*
313          * Let's get the other %cr4 stuff while we're here. Note, we defer
314          * enabling CR4_SMAP until startup_end(); however, that's importantly
315          * before we start other CPUs. That ensures that it will be synced out
316          * to other CPUs.
317          */
318         if (is_x86_feature(x86_featureset, X86FSET_DE))
319                 setcr4(getcr4() | CR4_DE);

321         if (is_x86_feature(x86_featureset, X86FSET_SMEP))
322                 setcr4(getcr4() | CR4_SMEP);
```



```
323 #endif /* __xpv */

325         /*
326          * initialize t0
327          */
328         t0.t_stk = (caddr_t)rp - MINFRAME;
329         t0.t_stkbase = t0stack;
330         t0.t_pri = maxclsyspri - 3;
331         t0.t_schedflag = TS_LOAD | TS_DONT_SWAP;
332         t0.t_procp = &p0;
333         t0.t_plockp = &p0lock.pl_lock;
334         t0.t_lwp = &lwp0;
335         t0.t_forw = &t0;
336         t0.t_back = &t0;
337         t0.t_next = &t0;
338         t0.t_prev = &t0;
339         t0.t_cpu = cpu[0];
340         t0.t_disp_queue = &cpu0_disp;
341         t0.t_bind_cpu = PBIND_NONE;
342         t0.t_bind_pset = PS_NONE;
343         t0.t_bindflag = (uchar_t)default_binding_mode;
344         t0.t_cpupart = &cp_default;
345         t0.t_clfuncs = &sys_classfuncs.thread;
346         t0.t_copyops = NULL;
347         THREAD_ONPROC(&t0, CPU);

349         lwp0.lwp_thread = &t0;
350         lwp0.lwp_regs = (void *)rp;
351         lwp0.lwp_procp = &p0;
352         t0.t_tid = p0.p_lwpcnt = p0.p_lwprcnt = p0.p_lwpid = 1;

354         p0.p_exec = NULL;
355         p0.p_stat = SRUN;
356         p0.p_flag = SSYS;
357         p0.p_tlist = &t0;
358         p0.p_stksize = 2*PAGESIZE;
359         p0.p_stkpageszc = 0;
360         p0.p_as = &kas;
361         p0.p_lockp = &p0lock;
362         p0.p_brkpageszc = 0;
363         p0.p_t1_lgrpid = LGRP_NONE;
364         p0.p_tr_lgrpid = LGRP_NONE;
365         psecflags_default(&p0.p_secflags);

367         sigorset(&p0.p_ignore, &ignoredefault);

369         CPU->cpu_thread = &t0;
370         bzero(&cpu0_disp, sizeof (disp_t));
371         CPU->cpu_disp = &cpu0_disp;
372         CPU->cpu_disp->disp_cpu = CPU;
373         CPU->cpu_dispthread = &t0;
374         CPU->cpu_idle_thread = &t0;
375         CPU->cpu_flags = CPU_READY | CPU_RUNNING | CPU_EXISTS | CPU_ENABLE;
376         CPU->cpu_dispatch_pri = t0.t_pri;

378         CPU->cpu_id = 0;

380         CPU->cpu_pri = 12;              /* initial PIL for the boot CPU */

382         /*
383          * Initialize thread/cpu microstate accounting
384          */
385         init_mstate(&t0, LMS_SYSTEM);
386         init_cpu_mstate(CPU, CMS_SYSTEM);

388         /*
```

```
389          * Initialize lists of available and active CPUs.
390          */
391         cpu_list_init(CPU);

393         pg_cpu_bootstrap(CPU);

395         /*
396          * Now that we have taken over the GDT, IDT and have initialized
397          * active CPU list it's time to inform kmdb if present.
398          */
399         if (boothowto & RB_DEBUG)
400                 kdi_idt_sync();

402         if (BOP_GETPROPLEN(bootops, "efi-systab") < 0) {
403                 /*
404                  * In BIOS system, explicitly set console to text mode (0x3)
405                  * if this is a boot post Fast Reboot, and the console is set
406                  * to CONS_SCREEN_TEXT.
407                  */
408                 if (post_fastreboot &&
409                     boot_console_type(NULL) == CONS_SCREEN_TEXT) {
410                         set_console_mode(0x3);
411                 }
412         }

414         /*
415          * If requested (boot -d) drop into kmdb.
416          *
417          * This must be done after cpu_list_init() on the 64-bit kernel
418          * since taking a trap requires that we re-compute gsbase based
419          * on the cpu list.
420          */
421         if (boothowto & RB_DEBUGENTER)
422                 kmdb_enter();

424         cpu_vm_data_init(CPU);

426         rp->r_fp = 0;   /* terminate kernel stack traces! */

428         prom_init("kernel", (void *)NULL);

430         /* User-set option overrides firmware value. */
431         if (bootprop_getval(PLAT_DR_OPTIONS_NAME, &prop_value) == 0) {
432                 plat_dr_options = (uint64_t)prop_value;
433         }
434 #if defined(__xpv)
435         /* No support of DR operations on xpv */
436         plat_dr_options = 0;
437 #else   /* __xpv */
438         /* Flag PLAT_DR_FEATURE_ENABLED should only be set by DR driver. */
439         plat_dr_options &= ~PLAT_DR_FEATURE_ENABLED;
440 #ifndef __amd64
441         /* Only enable CPU/memory DR on 64 bits kernel. */
442         plat_dr_options &= ~PLAT_DR_FEATURE_MEMORY;
443         plat_dr_options &= ~PLAT_DR_FEATURE_CPU;
444 #endif  /* __amd64 */
445 #endif  /* __xpv */

447         /*
448          * Get value of "plat_dr_physmax" boot option.
449          * It overrides values calculated from MSCT or SRAT table.
450          */
451         if (bootprop_getval(PLAT_DR_PHYSMAX_NAME, &prop_value) == 0) {
452                 plat_dr_physmax = ((uint64_t)prop_value) >> PAGESHIFT;
453         }
```



```
455         /* Get value of boot_ncpus. */
456         if (bootprop_getval(BOOT_NCPUS_NAME, &prop_value) != 0) {
457                 boot_ncpus = NCPU;
458         } else {
459                 boot_ncpus = (int)prop_value;
460                 if (boot_ncpus <= 0 || boot_ncpus > NCPU)
461                         boot_ncpus = NCPU;
462         }

464         /*
465          * Set max_ncpus and boot_max_ncpus to boot_ncpus if platform doesn't
466          * support CPU DR operations.
467          */
468         if (plat_dr_support_cpu() == 0) {
469                 max_ncpus = boot_max_ncpus = boot_ncpus;
470         } else {
471                 if (bootprop_getval(PLAT_MAX_NCPUS_NAME, &prop_value) != 0) {
472                         max_ncpus = NCPU;
473                 } else {
474                         max_ncpus = (int)prop_value;
475                         if (max_ncpus <= 0 || max_ncpus > NCPU) {
476                                 max_ncpus = NCPU;
477                         }
478                         if (boot_ncpus > max_ncpus) {
479                                 boot_ncpus = max_ncpus;
480                         }
481                 }

483                 if (bootprop_getval(BOOT_MAX_NCPUS_NAME, &prop_value) != 0) {
484                         boot_max_ncpus = boot_ncpus;
485                 } else {
486                         boot_max_ncpus = (int)prop_value;
487                         if (boot_max_ncpus <= 0 || boot_max_ncpus > NCPU) {
488                                 boot_max_ncpus = boot_ncpus;
489                         } else if (boot_max_ncpus > max_ncpus) {
490                                 boot_max_ncpus = max_ncpus;
491                         }
492                 }
493         }

495         /*
496          * Initialize the lgrp framework
497          */
498         lgrp_init(LGRP_INIT_STAGE1);

500         if (boothowto & RB_HALT) {
501                 prom_printf("unix: kernel halted by -h flag\n");
502                 prom_enter_mon();
503         }

505         ASSERT_STACK_ALIGNED();

507         /*
508          * Fill out cpu_ucode_info.  Update microcode if necessary.
509          */
510         ucode_check(CPU);
511         cpuid_pass_ucode(CPU, x86_featureset);

513         if (workaround_errata(CPU) != 0)
514                 panic("critical workaround(s) missing for boot cpu");
515 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    6486 Sat Mar 30 14:54:02 2019
new/usr/src/uts/i86pc/sys/apic_common.h
10597 would like a way to set NMI behavior at boot
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Andy Fiddaman <andy@omniosce.org>
**********************************************************
```

```
/*
 * 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) 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2017 by Delphix. All rights reserved.
 */
/*
 * Copyright 2019, Joyent, Inc.
 * Copyright 2018 Joyent, Inc.
 */

#ifndef _SYS_APIC_COMMON_H
#define _SYS_APIC_COMMON_H

#include <sys/psm_types.h>
#include <sys/avintr.h>
#include <sys/privregs.h>
#include <sys/pci.h>
#include <sys/cyclic.h>

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * Functions & Variables common to pcplusmp & apix
 */

#include <sys/psm_common.h>

/* Methods for multiple IOAPIC */
enum apic_ioapic_method_type {
        APIC_MUL_IOAPIC_NONE,           /* use to disable pcplusmp fallback */
        APIC_MUL_IOAPIC_MASK,           /* Set RT Entry Mask bit before EOI */
        APIC_MUL_IOAPIC_DEOI,           /* Directed EOI */
        APIC_MUL_IOAPIC_IOXAPIC,        /* IOxAPIC */
        APIC_MUL_IOAPIC_IIR,            /* IOMMU interrup remapping */
        APIC_MUL_IOAPIC_PCPLUSMP        /* Fall back to old pcplusmp */
};

#define APIX_IS_DIRECTED_EOI(type)      \
```



```
        ((type) == APIC_MUL_IOAPIC_DEOI || (type) == APIC_MUL_IOAPIC_IIR)
#define APIX_IS_MASK_RDT(type)  \
        ((type) == APIC_MUL_IOAPIC_NONE || (type) == APIC_MUL_IOAPIC_MASK)

extern int      apix_enable;
extern int      apix_loaded(void);
extern enum apic_ioapic_method_type apix_mul_ioapic_method;

extern int      apic_oneshot;
/* to allow disabling one-shot capability */
extern int      apic_oneshot_enable;

/* Now the ones for Dynamic Interrupt distribution */
extern int      apic_enable_dynamic_migration;

extern int apic_have_32bit_cr8;

extern struct psm_ops *psmops;

/*
 * These variables are frequently accessed in apic_intr_enter(),
 * apic_intr_exit and apic_setspl, so group them together
 */
extern volatile uint32_t *apicadr;      /* virtual addr of local APIC   */
extern uchar_t  apic_io_vectbase[MAX_IO_APIC];
extern uchar_t  apic_io_vectend[MAX_IO_APIC];
extern uchar_t  apic_io_ver[MAX_IO_APIC];
extern int      apic_io_max;
extern int      apic_nvidia_io_max;
extern int apic_setspl_delay;           /* apic_setspl - delay enable   */
extern int apic_clkvect;

/* vector at which error interrupts come in */
extern int apic_errvect;
extern int apic_enable_error_intr;
extern int apic_error_display_delay;

/* vector at which performance counter overflow interrupts come in */
extern int apic_cpcovf_vect;
extern int apic_enable_cpcovf_intr;

/* vector at which CMCI interrupts come in */
extern int apic_cmci_vect;
extern int cmi_enable_cmci;
extern void cmi_cmci_trap(void);

extern kmutex_t cmci_cpu_setup_lock;    /* protects cmci_cpu_setup_registered */
extern int cmci_cpu_setup_registered;

extern int      apic_forceload;

extern int      apic_coarse_hrtime;     /* 0 - use accurate slow gethrtime() */
                                        /* 1 - use gettime() for performance */
extern int      apic_flat_model;                /* 0 - clustered. 1 - flat */

extern int      apic_panic_on_nmi;
extern int      apic_panic_on_apic_error;

extern int      apic_verbose;

extern int      apic_pir_vect;

#ifdef DEBUG
extern int      apic_debug;
extern int      apic_restrict_vector;
```

```
125 extern int      apic_debug_msgbuf[APIC_DEBUG_MSGBUFSIZE];
126 extern int      apic_debug_msgbufindex;

128 #endif /* DEBUG */

130 extern uint_t   apic_nsec_per_intr;
131 extern uint_t   apic_nticks;
132 extern uint_t   apic_skipped_redistribute;

134 extern uint_t   last_count_read;
135 extern lock_t   apic_mode_switch_lock;
136 extern lock_t   apic_gethrtime_lock;
137 extern volatile int     apic_hrtime_stamp;
138 extern volatile hrtime_t apic_nsec_since_boot;
139 extern uint_t   apic_hertz_count;

141 extern uint64_t apic_ticks_per_SFnsecs; /* # of ticks in SF nsecs */

143 extern int      apic_hrtime_error;
144 extern int      apic_remote_hrterr;
145 extern int      apic_num_nmis;
146 extern int      apic_apic_error;
147 extern int      apic_num_apic_errors;
148 extern int      apic_num_cksum_errors;

150 extern int      apic_error;

152 /* use to make sure only one cpu handles the nmi */
153 extern lock_t   apic_nmi_lock;
154 /* use to make sure only one cpu handles the error interrupt */
155 extern lock_t   apic_error_lock;

157 /* Patchable global variables. */
158 extern int      apic_kmdb_on_nmi;       /* 0 - no, 1 - yes enter kmdb */
158 extern uint32_t apic_divide_reg_init;   /* 0 - divide by 2 */

160 extern apic_intrmap_ops_t *apic_vt_ops;

162 #ifdef  DEBUG
163 extern int      apic_break_on_cpu;
164 extern int      apic_stretch_interrupts;
165 extern int      apic_stretch_ISR;       /* IPL of 3 matches nothing now */
166 #endif

168 extern cyclic_id_t apic_cyclic_id;

170 extern void apic_nmi_intr(caddr_t arg, struct regs *rp);
171 extern int      apic_clkinit();
172 extern hrtime_t apic_gettime();
173 extern hrtime_t apic_gethrtime();
174 extern int      apic_cpu_start(processorid_t cpuid, caddr_t ctx);
175 extern int      apic_cpu_stop(processorid_t cpuid, caddr_t ctx);
176 extern int      apic_cpu_add(psm_cpu_request_t *reqp);
177 extern int      apic_cpu_remove(psm_cpu_request_t *reqp);
178 extern int      apic_cpu_ops(psm_cpu_request_t *reqp);
179 extern void     apic_switch_ipi_callback(boolean_t enter);
180 extern void     apic_send_ipi(int cpun, int ipl);
181 extern void     apic_set_idlecpu(processorid_t cpun);
182 extern void     apic_unset_idlecpu(processorid_t cpun);
183 extern void     apic_shutdown(int cmd, int fcn);
184 extern void     apic_preshutdown(int cmd, int fcn);
185 extern processorid_t    apic_get_next_processorid(processorid_t cpun);
186 extern uint64_t apic_calibrate();
187 extern int      apic_get_pir_ipivect(void);
188 extern void     apic_send_pir_ipi(processorid_t);
```



```
190 extern int apic_error_intr();
191 extern void apic_cpcovf_mask_clear(void);
192 extern void apic_cmci_setup(processorid_t, boolean_t);
193 extern void apic_intrmap_init(int apic_mode);
194 extern processorid_t apic_find_cpu(int flag);
195 extern processorid_t apic_get_next_bind_cpu(void);

197 extern int      apic_support_msi;
198 extern int      apic_multi_msi_enable;
199 extern int      apic_msix_enable;

201 extern uint32_t apic_get_localapicid(uint32_t cpuid);
202 extern uchar_t apic_get_ioapicid(uchar_t ioapicindex);

204 typedef enum nmi_action {
205         NMI_ACTION_UNSET,
206         NMI_ACTION_PANIC,
207         NMI_ACTION_IGNORE,
208         NMI_ACTION_KMDB
209 } nmi_action_t;

211 extern nmi_action_t nmi_action;

213 #ifdef  __cplusplus
214 }
```

*unchanged_portion_omitted_*

```
**********************************************************
    7334 Sat Mar 30 14:54:03 2019
new/usr/src/uts/intel/sys/bootconf.h
10597 would like a way to set NMI behavior at boot
Reviewed by: Robert Mustacchi <rm@joyent.com>
Reviewed by: Andy Fiddaman <andy@omniosce.org>
**********************************************************
```

```
/*
 * 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 2016 Nexenta Systems, Inc.
 * Copyright 2019, Joyent, Inc.
 */

#ifndef _SYS_BOOTCONF_H
#define _SYS_BOOTCONF_H


/*
 * Boot time configuration information objects
 */

#include <sys/types.h>
#include <sys/bootregs.h>               /* for struct bop_regs */
#include <sys/bootstat.h>
#include <sys/dirent.h>                 /* for struct dirent */
#include <sys/memlist.h>
#include <sys/obpdefs.h>
#include <sys/varargs.h>
#include <net/if.h>                     /* for IFNAMSIZ */

#ifdef __cplusplus
extern "C" {
#endif

#define BP_MAX_STRLEN   32

/*
 * Boot property names
 */
#define BP_CPU_APICID_ARRAY     "cpu_apicid_array"
#define BP_LGRP_SLIT_ENABLE     "lgrp_slit_enable"
#define BP_LGRP_SRAT_ENABLE     "lgrp_srat_enable"
#define BP_LGRP_MSCT_ENABLE     "lgrp_msct_enable"
#define BP_LGRP_TOPO_LEVELS     "lgrp_topo_levels"
```



```
/*
 * masks to hand to bsys_alloc memory allocator
 * XXX  These names shouldn't really be srmmu derived.
 */
#define BO_NO_ALIGN     0x00001000

/* flags for BOP_EALLOC */
#define BOPF_X86_ALLOC_CLIENT   0x001
#define BOPF_X86_ALLOC_REAL     0x002
#define BOPF_X86_ALLOC_IDMAP    0x003
#define BOPF_X86_ALLOC_PHYS     0x004

/* return values for the newer bootops */
#define BOOT_SUCCESS    0
#define BOOT_FAILURE    (-1)

/* top of boot scratch memory: 15 MB; multiboot loads at 16 MB */
#define MAGIC_PHYS      0xF00000

/*
 *  We pass a ptr to the space that boot has been using
 *  for its memory lists.
 */
struct bsys_mem {
        struct memlist  *physinstalled; /* amt of physmem installed */
        struct memlist  *rsvdmem;       /* amt of bios reserved mem */
        struct memlist  *physavail;     /* amt of physmem avail for use */
        struct memlist  *virtavail;     /* amt of virtmem avail for use */
        struct memlist  *pcimem;        /* amt of pcimem avail for use */
        uint_t          extent;         /* number of bytes in the space */
};
______unchanged_portion_omitted_

/*
 * flags
 */
#define BO_VALID        0x01    /* all information in object is valid */
#define BO_BUSY         0x02    /* object is busy */

extern struct bootobj rootfs;
extern struct bootobj swapfile;

extern char obp_bootpath[BO_MAXOBJNAME];

extern void *gfx_devinfo_list;

extern dev_t getrootdev(void);
extern void getfsname(char *, char *, size_t);
extern int loadrootmodules(void);

extern int strplumb(void);
extern int strplumb_load(void);
extern char *strplumb_get_netdev_path(void);

extern void consconfig(void);
extern void release_bootstrap(void);

extern void param_check(void);
extern int octet_to_hexascii(const void *, uint_t, char *, uint_t *);

extern int dhcpinit(void);

extern struct bootops *bootops;
extern int netboot;
extern int swaploaded;
```

```
224 extern int modrootloaded;
225 extern char kern_bootargs[];
226 extern char kern_bootfile[];
227 extern char *kobj_module_path;
228 extern char *default_path;
229 extern char *dhcack;
230 extern int dhcacklen;
231 extern char dhcifname[IFNAMSIZ];
232 extern char *netdev_path;

234 extern void bop_no_more_mem(void);

236 /*PRINTFLIKE2*/
237 extern void bop_printf(void *, const char *, ...)
238     __KPRINTFLIKE(2);
239 extern void vbop_printf(void *, const char *, va_list);

241 /*PRINTFLIKE1*/
242 extern void bop_panic(const char *, ...)
243     __KPRINTFLIKE(1) __NORETURN;
244 #pragma rarely_called(bop_panic)

246 extern void boot_prop_finish(void);

248 extern int bootprop_getval(const char *, u_longlong_t *);
249 extern int bootprop_getstr(const char *, char *, size_t);

251 /*
252  * Back door to fakebop.c to get physical memory allocated.
253  * 64 bit data types are fixed for 32 bit PAE use.
254  */
255 extern paddr_t do_bop_phys_alloc(uint64_t, uint64_t);

257 extern int do_bsys_getproplen(bootops_t *, const char *);
258 extern int do_bsys_getprop(bootops_t *, const char *, void *);
259 extern int do_bsys_getproptype(bootops_t *, const char *);

261 #endif /* _KERNEL && !_BOOT */

263 #ifdef __cplusplus
264 }
```

*unchanged_portion_omitted_*