```
**********************************************************
   12035 Wed Jan 17 15:42:49 2018
new/usr/src/uts/i86pc/io/todpc_subr.c
8680 Time of Day clock error
**********************************************************
/*
 * 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 2018 Gary Mills
 * Copyright 2012 Nexenta Systems, Inc.  All rights reserved.
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*      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/time.h>
#include <sys/systm.h>

#include <sys/cpuvar.h>
#include <sys/clock.h>
#include <sys/debug.h>
#include <sys/rtc.h>
#include <sys/archsystm.h>
#include <sys/sysmacros.h>
#include <sys/lockstat.h>
#include <sys/stat.h>
#include <sys/sunddi.h>
#include <sys/ddi.h>

#include <sys/acpi/acpi.h>
#include <sys/acpica.h>

static int todpc_rtcget(unsigned char *buf);
static void todpc_rtcput(unsigned char *buf);

#define CLOCK_RES       1000            /* 1 microsec in nanosecs */

int clock_res = CLOCK_RES;
```

```
/*
 * The minimum sleep time till an alarm can be fired.
 * This can be tuned in /etc/system, but if the value is too small,
 * there is a danger that it will be missed if it takes too long to
 * get from the set point to sleep.  Or that it can fire quickly, and
 * generate a power spike on the hardware.  And small values are
 * probably only usefull for test setups.
 */
int clock_min_alarm = 4;

/*
 * Machine-dependent clock routines.
 */

extern long gmt_lag;

struct rtc_offset {
        int8_t  loaded;
        uint8_t day_alrm;
        uint8_t mon_alrm;
        uint8_t century;
};
______unchanged_portion_omitted_

/*
 * Routine to read contents of real time clock to the specified buffer.
 * Returns ENXIO if clock not valid, or EAGAIN if clock data cannot be read
 * else 0.
 * Some RTC hardware is very slow at asserting the validity flag on
 * startup.  The routine will busy wait for the RTC to become valid.
 * The routine will also busy wait for the Update-In-Progress flag to clear.
 * The routine will busy wait for the Update-In-Progress flag to clear.
 * On completion of the reads the Seconds register is re-read and the
 * UIP flag is rechecked to confirm that an clock update did not occur
 * during the accesses.  Routine will error exit after 256 attempts.
 * (See bugid 1158298.)
 * Routine returns RTC_NREG (which is 15) bytes of data, as given in the
 * technical reference.  This data includes both time and status registers.
 */

static int
todpc_rtcget(unsigned char *buf)
{
        unsigned char   reg;
        int             i;
        int             uip_try = 256;
        int             vrt_try = 512;
        int             retries = 256;
        unsigned char   *rawp;
        unsigned char   century = RTC_CENTURY;
        unsigned char   day_alrm;
        unsigned char   mon_alrm;

        ASSERT(MUTEX_HELD(&tod_lock));

        day_alrm = pc_rtc_offset.day_alrm;
        mon_alrm = pc_rtc_offset.mon_alrm;
        if (pc_rtc_offset.century != 0) {
                century = pc_rtc_offset.century;
        }

        for (;;) {
                if (vrt_try-- < 0)
                        return (ENXIO);
                outb(RTC_ADDR, RTC_D);          /* check if clock valid */
                reg = inb(RTC_DATA);
                if ((reg & RTC_VRT) != 0)
                        break;
                drv_usecwait(5000);             /* Delay for 5000 us */
        }
        if ((reg & RTC_VRT) == 0)
                return (ENXIO);


checkuip:
        if (uip_try-- < 0)
        if (retries-- < 0)
                return (EAGAIN);
        outb(RTC_ADDR, RTC_A);          /* check if update in progress */
        reg = inb(RTC_DATA);
        if (reg & RTC_UIP) {
                tenmicrosec();
                goto checkuip;
        }

        for (i = 0, rawp = buf; i < RTC_NREG; i++) {
                outb(RTC_ADDR, i);
                *rawp++ = inb(RTC_DATA);
        }
        outb(RTC_ADDR, century); /* do century */
        ((struct rtc_t *)buf)->rtc_century = inb(RTC_DATA);

        if (day_alrm > 0) {
                outb(RTC_ADDR, day_alrm);
                ((struct rtc_t *)buf)->rtc_adom = inb(RTC_DATA) & 0x3f;
        }
        if (mon_alrm > 0) {
                outb(RTC_ADDR, mon_alrm);
                ((struct rtc_t *)buf)->rtc_amon = inb(RTC_DATA);
        }

        outb(RTC_ADDR, 0);              /* re-read Seconds register */
        reg = inb(RTC_DATA);
        if (reg != ((struct rtc_t *)buf)->rtc_sec ||
            (((struct rtc_t *)buf)->rtc_statusa & RTC_UIP))
                /* update occured during reads */
                goto checkuip;

        return (0);
}
```
______***unchanged_portion_omitted_***