```
**********************************************************
   35452 Tue Apr 16 05:23:03 2019
new/usr/src/lib/libefi/common/rdwr_efi.c
10570 Need workaround to EFI boot on AMI BIOS
**********************************************************
/*
 * 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) 2002, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
 * Copyright 2014 Toomas Soome <tsoome@me.com>
 * Copyright 2018 OmniOS Community Edition (OmniOSce) Association.
 * Copyright 2019 Joyent, Inc.
 * Copyright (c) 2018, Joyent, Inc.
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <strings.h>
#include <unistd.h>
#include <smbios.h>
#include <uuid/uuid.h>
#include <libintl.h>
#include <sys/types.h>
#include <sys/dkio.h>
#include <sys/vtoc.h>
#include <sys/mhd.h>
#include <sys/param.h>
#include <sys/dktp/fdisk.h>
#include <sys/efi_partition.h>
#include <sys/byteorder.h>
#include <sys/ddi.h>

/*
 * The original conversion array used simple array index, but since
 * we do need to take account of VTOC tag numbers from other systems,
 * we need to provide tag values too, or the array will grow too large.
 *
 * Still we will fabricate the missing p_tag values.
 */
static struct uuid_to_ptag {
        struct uuid     uuid;
        ushort_t        p_tag;
} conversion_array[] = {
______unchanged_portion_omitted_
```

```
static int
check_label(int fd, dk_efi_t *dk_ioc)
{
        efi_gpt_t               *efi;
        uint_t                  crc;

        if (efi_ioctl(fd, DKIOCGETEFI, dk_ioc) == -1) {
                switch (errno) {
                case EIO:
                        return (VT_EIO);
                default:
                        return (VT_ERROR);
                }
        }
        efi = dk_ioc->dki_data;
        if (efi->efi_gpt_Signature != LE_64(EFI_SIGNATURE)) {
                if (efi_debug)
                        (void) fprintf(stderr,
                            "Bad EFI signature: 0x%llx != 0x%llx\n",
                            (long long)efi->efi_gpt_Signature,
                            (long long)LE_64(EFI_SIGNATURE));
                return (VT_EINVAL);
        }

        /*
         * check CRC of the header; the size of the header should
         * never be larger than one block
         */
        crc = efi->efi_gpt_HeaderCRC32;
        efi->efi_gpt_HeaderCRC32 = 0;

        if (((len_t)LE_32(efi->efi_gpt_HeaderSize) > dk_ioc->dki_length) ||
            crc != LE_32(efi_crc32((unsigned char *)efi,
            LE_32(efi->efi_gpt_HeaderSize)))) {
                if (efi_debug)
                        (void) fprintf(stderr,
                            "Bad EFI CRC: 0x%x != 0x%x\n",
                            crc, LE_32(efi_crc32((unsigned char *)efi,
                            LE_32(efi->efi_gpt_HeaderSize))));
                            crc,
                            LE_32(efi_crc32((unsigned char *)efi,
                            sizeof (struct efi_gpt))));
                return (VT_EINVAL);
        }

        return (0);
}
______unchanged_portion_omitted_

/* writes a "protective" MBR */
static int
write_pmbr(int fd, struct dk_gpt *vtoc)
{
        dk_efi_t        dk_ioc;
        struct mboot    mb;
        uchar_t         *cp;
        diskaddr_t      size_in_lba;
        uchar_t         *buf;
        int             len, slot, active;

        slot = active = 0;

        hardware_workarounds(&slot, &active);

        len = (vtoc->efi_lbasize == 0) ? sizeof (mb) : vtoc->efi_lbasize;
        buf = calloc(1, len);
        buf = calloc(len, 1);

        /*
         * Preserve any boot code and disk signature if the first block is
         * already an MBR.
         */
        dk_ioc.dki_lba = 0;
        dk_ioc.dki_length = len;
        /* LINTED -- always longlong aligned */
        dk_ioc.dki_data = (efi_gpt_t *)buf;
        if (efi_ioctl(fd, DKIOCGETEFI, &dk_ioc) == -1) {
                (void) memcpy(&mb, buf, sizeof (mb));
                bzero(&mb, sizeof (mb));
                mb.signature = LE_16(MBB_MAGIC);
        } else {
                (void) memcpy(&mb, buf, sizeof (mb));
                if (mb.signature != LE_16(MBB_MAGIC)) {
                        bzero(&mb, sizeof (mb));
                        mb.signature = LE_16(MBB_MAGIC);
                }
        }

        bzero(&mb.parts, sizeof (mb.parts));
        cp = (uchar_t *)&mb.parts[slot * sizeof (struct ipart)];
        /* bootable or not */
        *cp++ = active ? ACTIVE : NOTACTIVE;
        /* beginning CHS; same as starting LBA (but one-based) */
        *cp++ = 0x0;
        *cp++ = 0x2;
        *cp++ = 0x0;
        /* beginning CHS; 0xffffff if not representable */
        *cp++ = 0xff;
        *cp++ = 0xff;
        *cp++ = 0xff;
        /* OS type */
        *cp++ = EFI_PMBR;
        /* ending CHS; 0xffffff if not representable */
        *cp++ = 0xff;
        *cp++ = 0xff;
        *cp++ = 0xff;
        /* starting LBA: 1 (little endian format) by EFI definition */
        *cp++ = 0x01;
        *cp++ = 0x00;
        *cp++ = 0x00;
        *cp++ = 0x00;
        /* ending LBA: last block on the disk (little endian format) */
        size_in_lba = vtoc->efi_last_lba;
        if (size_in_lba < 0xffffffff) {
                *cp++ = (size_in_lba & 0x000000ff);
                *cp++ = (size_in_lba & 0x0000ff00) >> 8;
                *cp++ = (size_in_lba & 0x00ff0000) >> 16;
                *cp++ = (size_in_lba & 0xff000000) >> 24;
        } else {
                *cp++ = 0xff;
                *cp++ = 0xff;
                *cp++ = 0xff;
                *cp++ = 0xff;
        }

        (void) memcpy(buf, &mb, sizeof (mb));
        /* LINTED -- always longlong aligned */
        dk_ioc.dki_data = (efi_gpt_t *)buf;
        dk_ioc.dki_lba = 0;
        dk_ioc.dki_length = len;
        if (efi_ioctl(fd, DKIOCSETEFI, &dk_ioc) == -1) {
                free(buf);
```

```
                switch (errno) {
                case EIO:
                        return (VT_EIO);
                case EINVAL:
                        return (VT_EINVAL);
                default:
                        return (VT_ERROR);
                }
        }
        free(buf);
        return (0);
}
```

______***unchanged_portion_omitted_***

```
/*
 * write EFI label and backup label
 */
int
efi_write(int fd, struct dk_gpt *vtoc)
{
        dk_efi_t                dk_ioc;
        efi_gpt_t               *efi;
        efi_gpe_t               *efi_parts;
        int                     i, j;
        struct dk_cinfo         dki_info;
        int                     nblocks;
        diskaddr_t              lba_backup_gpt_hdr;

        if (ioctl(fd, DKIOCINFO, (caddr_t)&dki_info) == -1) {
                if (efi_debug)
                        (void) fprintf(stderr, "DKIOCINFO errno 0x%x\n", errno);
                switch (errno) {
                case EIO:
                        return (VT_EIO);
                case EINVAL:
                        return (VT_EINVAL);
                default:
                        return (VT_ERROR);
                }
        }

        if (check_input(vtoc))
                return (VT_EINVAL);

        dk_ioc.dki_lba = 1;
        if (NBLOCKS(vtoc->efi_nparts, vtoc->efi_lbasize) < 34) {
                dk_ioc.dki_length = EFI_MIN_ARRAY_SIZE + vtoc->efi_lbasize;
        } else {
                dk_ioc.dki_length = NBLOCKS(vtoc->efi_nparts,
                    vtoc->efi_lbasize) *
                    vtoc->efi_lbasize;
        }

        /*
         * the number of blocks occupied by GUID partition entry array
         */
        nblocks = dk_ioc.dki_length / vtoc->efi_lbasize - 1;

        /*
         * Backup GPT header is located on the block after GUID
         * partition entry array. Here, we calculate the address
         * for backup GPT header.
         */
        lba_backup_gpt_hdr = vtoc->efi_last_u_lba + 1 + nblocks;
        if ((dk_ioc.dki_data = calloc(1, dk_ioc.dki_length)) == NULL)
```

```c
                return (VT_ERROR);

        efi = dk_ioc.dki_data;

        /* stuff user's input into EFI struct */
        efi->efi_gpt_Signature = LE_64(EFI_SIGNATURE);
        efi->efi_gpt_Revision = LE_32(vtoc->efi_version); /* 0x02000100 */
        efi->efi_gpt_HeaderSize = LE_32(EFI_HEADER_SIZE);
        efi->efi_gpt_HeaderSize = LE_32(sizeof (struct efi_gpt));
        efi->efi_gpt_Reserved1 = 0;
        efi->efi_gpt_MyLBA = LE_64(1ULL);
        efi->efi_gpt_AlternateLBA = LE_64(lba_backup_gpt_hdr);
        efi->efi_gpt_FirstUsableLBA = LE_64(vtoc->efi_first_u_lba);
        efi->efi_gpt_LastUsableLBA = LE_64(vtoc->efi_last_u_lba);
        efi->efi_gpt_PartitionEntryLBA = LE_64(2ULL);
        efi->efi_gpt_NumberOfPartitionEntries = LE_32(vtoc->efi_nparts);
        efi->efi_gpt_SizeOfPartitionEntry = LE_32(sizeof (struct efi_gpe));
        UUID_LE_CONVERT(efi->efi_gpt_DiskGUID, vtoc->efi_disk_uguid);

        /* LINTED -- always longlong aligned */
        efi_parts = (efi_gpe_t *)((char *)dk_ioc.dki_data + vtoc->efi_lbasize);

        for (i = 0; i < vtoc->efi_nparts; i++) {
                for (j = 0;
                    j < sizeof (conversion_array) /
                    sizeof (struct uuid_to_ptag); j++) {

                        if (vtoc->efi_parts[i].p_tag ==
                            conversion_array[j].p_tag) {
                                UUID_LE_CONVERT(
                                    efi_parts[i].efi_gpe_PartitionTypeGUID,
                                    conversion_array[j].uuid);
                                break;
                        }
                }

                if (j == sizeof (conversion_array) /
                    sizeof (struct uuid_to_ptag)) {
                        /*
                         * If we didn't have a matching uuid match, bail here.
                         * Don't write a label with unknown uuid.
                         */
                        if (efi_debug) {
                                (void) fprintf(stderr,
                                    "Unknown uuid for p_tag %d\n",
                                    vtoc->efi_parts[i].p_tag);
                        }
                        return (VT_EINVAL);
                }

                efi_parts[i].efi_gpe_StartingLBA =
                    LE_64(vtoc->efi_parts[i].p_start);
                efi_parts[i].efi_gpe_EndingLBA =
                    LE_64(vtoc->efi_parts[i].p_start +
                    vtoc->efi_parts[i].p_size - 1);
                efi_parts[i].efi_gpe_Attributes.PartitionAttrs =
                    LE_16(vtoc->efi_parts[i].p_flag);
                for (j = 0; j < EFI_PART_NAME_LEN; j++) {
                        efi_parts[i].efi_gpe_PartitionName[j] =
                            LE_16((ushort_t)vtoc->efi_parts[i].p_name[j]);
                }
                if ((vtoc->efi_parts[i].p_tag != V_UNASSIGNED) &&
                    uuid_is_null((uchar_t *)&vtoc->efi_parts[i].p_uguid)) {
                        (void) uuid_generate((uchar_t *)
                            &vtoc->efi_parts[i].p_uguid);
                }
```

```c
                bcopy(&vtoc->efi_parts[i].p_uguid,
                    &efi_parts[i].efi_gpe_UniquePartitionGUID,
                    sizeof (uuid_t));
        }
        efi->efi_gpt_PartitionEntryArrayCRC32 =
            LE_32(efi_crc32((unsigned char *)efi_parts,
            vtoc->efi_nparts * (int)sizeof (struct efi_gpe)));
        efi->efi_gpt_HeaderCRC32 = LE_32(efi_crc32((unsigned char *)efi,
            EFI_HEADER_SIZE));
        efi->efi_gpt_HeaderCRC32 =
            LE_32(efi_crc32((unsigned char *)efi, sizeof (struct efi_gpt)));

        if (efi_ioctl(fd, DKIOCSETEFI, &dk_ioc) == -1) {
                free(dk_ioc.dki_data);
                switch (errno) {
                case EIO:
                        return (VT_EIO);
                case EINVAL:
                        return (VT_EINVAL);
                default:
                        return (VT_ERROR);
                }
        }

        /* write backup partition array */
        dk_ioc.dki_lba = vtoc->efi_last_u_lba + 1;
        dk_ioc.dki_length -= vtoc->efi_lbasize;
        /* LINTED */
        dk_ioc.dki_data = (efi_gpt_t *)((char *)dk_ioc.dki_data +
            vtoc->efi_lbasize);

        if (efi_ioctl(fd, DKIOCSETEFI, &dk_ioc) == -1) {
                /*
                 * we wrote the primary label okay, so don't fail
                 */
                if (efi_debug) {
                        (void) fprintf(stderr,
                            "write of backup partitions to block %llu "
                            "failed, errno %d\n",
                            vtoc->efi_last_u_lba + 1,
                            errno);
                }
        }
        /*
         * now swap MyLBA and AlternateLBA fields and write backup
         * partition table header
         */
        dk_ioc.dki_lba = lba_backup_gpt_hdr;
        dk_ioc.dki_length = vtoc->efi_lbasize;
        /* LINTED */
        dk_ioc.dki_data = (efi_gpt_t *)((char *)dk_ioc.dki_data -
            vtoc->efi_lbasize);
        efi->efi_gpt_AlternateLBA = LE_64(1ULL);
        efi->efi_gpt_MyLBA = LE_64(lba_backup_gpt_hdr);
        efi->efi_gpt_PartitionEntryLBA = LE_64(vtoc->efi_last_u_lba + 1);
        efi->efi_gpt_HeaderCRC32 = 0;
        efi->efi_gpt_HeaderCRC32 =
            LE_32(efi_crc32((unsigned char *)dk_ioc.dki_data, EFI_HEADER_SIZE));
            LE_32(efi_crc32((unsigned char *)dk_ioc.dki_data,
            sizeof (struct efi_gpt)));

        if (efi_ioctl(fd, DKIOCSETEFI, &dk_ioc) == -1) {
                if (efi_debug) {
                        (void) fprintf(stderr,
                            "write of backup header to block %llu failed, "
                            "errno %d\n",
                            lba_backup_gpt_hdr,
                            errno);
                }
        }
        /* write the PMBR */
        (void) write_pmbr(fd, vtoc);
        free(dk_ioc.dki_data);
        return (0);
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
   54292 Tue Apr 16 05:23:05 2019
new/usr/src/uts/common/fs/zfs/zvol.c
10570 Need workaround to EFI boot on AMI BIOS
**********************************************************
/*
 * 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) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
 *
 * Portions Copyright 2010 Robert Milkowski
 *
 * Copyright 2017 Nexenta Systems, Inc.  All rights reserved.
 * Copyright (c) 2012, 2017 by Delphix. All rights reserved.
 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
 * Copyright (c) 2014 Integros [integros.com]
 * Copyright (c) 2019, Joyent, Inc.
 */

/*
 * ZFS volume emulation driver.
 *
 * Makes a DMU object look like a volume of arbitrary size, up to 2^64 bytes.
 * Volumes are accessed through the symbolic links named:
 *
 * /dev/zvol/dsk/<pool_name>/<dataset_name>
 * /dev/zvol/rdsk/<pool_name>/<dataset_name>
 *
 * These links are created by the /dev filesystem (sdev_zvolops.c).
 * Volumes are persistent through reboot.  No user command needs to be
 * run before opening and using a device.
 */

#include <sys/types.h>
#include <sys/param.h>
#include <sys/errno.h>
#include <sys/uio.h>
#include <sys/buf.h>
#include <sys/modctl.h>
#include <sys/open.h>
#include <sys/kmem.h>
#include <sys/conf.h>
#include <sys/cmn_err.h>
#include <sys/stat.h>
#include <sys/zap.h>
#include <sys/spa.h>
#include <sys/spa_impl.h>
#include <sys/zio.h>
```

```
#include <sys/dmu_traverse.h>
#include <sys/dnode.h>
#include <sys/dsl_dataset.h>
#include <sys/dsl_prop.h>
#include <sys/dkio.h>
#include <sys/efi_partition.h>
#include <sys/byteorder.h>
#include <sys/pathname.h>
#include <sys/ddi.h>
#include <sys/sunddi.h>
#include <sys/crc32.h>
#include <sys/dirent.h>
#include <sys/policy.h>
#include <sys/fs/zfs.h>
#include <sys/zfs_ioctl.h>
#include <sys/mkdev.h>
#include <sys/zil.h>
#include <sys/refcount.h>
#include <sys/zfs_znode.h>
#include <sys/zfs_rlock.h>
#include <sys/vdev_disk.h>
#include <sys/vdev_impl.h>
#include <sys/vdev_raidz.h>
#include <sys/zvol.h>
#include <sys/dumphdr.h>
#include <sys/zil_impl.h>
#include <sys/dbuf.h>
#include <sys/dmu_tx.h>
#include <sys/zfeature.h>
#include <sys/zio_checksum.h>
#include <sys/zil_impl.h>
#include <sys/dkioc_free_util.h>
#include <sys/zfs_rlock.h>

#include "zfs_namecheck.h"

void *zfsdev_state;
static char *zvol_tag = "zvol_tag";

#define ZVOL_DUMPSIZE           "dumpsize"

/*
 * This lock protects the zfsdev_state structure from being modified
 * while it's being used, e.g. an open that comes in before a create
 * finishes.  It also protects temporary opens of the dataset so that,
 * e.g., an open doesn't get a spurious EBUSY.
 */
kmutex_t zfsdev_state_lock;
static uint32_t zvol_minors;

typedef struct zvol_extent {
        list_node_t     ze_node;
        dva_t           ze_dva;         /* dva associated with this extent */
        uint64_t        ze_nblks;       /* number of blocks in extent */
} zvol_extent_t;
______unchanged_portion_omitted_

int
zvol_getefi(void *arg, int flag, uint64_t vs, uint8_t bs)
{
        struct uuid uuid = EFI_RESERVED;
        efi_gpe_t gpe = { 0 };
        uint32_t crc;
        dk_efi_t efi;
        int length;
        char *ptr;
        if (ddi_copyin(arg, &efi, sizeof (dk_efi_t), flag))
                return (SET_ERROR(EFAULT));
        ptr = (char *)(uintptr_t)efi.dki_data_64;
        length = efi.dki_length;
        /*
         * Some clients may attempt to request a PMBR for the
         * zvol.  Currently this interface will return EINVAL to
         * such requests.  These requests could be supported by
         * adding a check for lba == 0 and consing up an appropriate
         * PMBR.
         */
        if (efi.dki_lba < 1 || efi.dki_lba > 2 || length <= 0)
                return (SET_ERROR(EINVAL));

        gpe.efi_gpe_StartingLBA = LE_64(34ULL);
        gpe.efi_gpe_EndingLBA = LE_64((vs >> bs) - 1);
        UUID_LE_CONVERT(gpe.efi_gpe_PartitionTypeGUID, uuid);

        if (efi.dki_lba == 1) {
                efi_gpt_t gpt = { 0 };

                gpt.efi_gpt_Signature = LE_64(EFI_SIGNATURE);
                gpt.efi_gpt_Revision = LE_32(EFI_VERSION_CURRENT);
                gpt.efi_gpt_HeaderSize = LE_32(EFI_HEADER_SIZE);
                gpt.efi_gpt_HeaderSize = LE_32(sizeof (gpt));
                gpt.efi_gpt_MyLBA = LE_64(1ULL);
                gpt.efi_gpt_FirstUsableLBA = LE_64(34ULL);
                gpt.efi_gpt_LastUsableLBA = LE_64((vs >> bs) - 1);
                gpt.efi_gpt_PartitionEntryLBA = LE_64(2ULL);
                gpt.efi_gpt_NumberOfPartitionEntries = LE_32(1);
                gpt.efi_gpt_SizeOfPartitionEntry =
                    LE_32(sizeof (efi_gpe_t));
                CRC32(crc, &gpe, sizeof (gpe), -1U, crc32_table);
                gpt.efi_gpt_PartitionEntryArrayCRC32 = LE_32(~crc);
                CRC32(crc, &gpt, EFI_HEADER_SIZE, -1U, crc32_table);
                CRC32(crc, &gpt, sizeof (gpt), -1U, crc32_table);
                gpt.efi_gpt_HeaderCRC32 = LE_32(~crc);
                if (ddi_copyout(&gpt, ptr, MIN(sizeof (gpt), length),
                    flag))
                        return (SET_ERROR(EFAULT));
                ptr += sizeof (gpt);
                length -= sizeof (gpt);
        }
        if (length > 0 && ddi_copyout(&gpe, ptr, MIN(sizeof (gpe),
            length), flag))
                return (SET_ERROR(EFAULT));
        return (0);
}
```
______***unchanged_portion_omitted_***

```
***********************************************************
  156604 Tue Apr 16 05:23:06 2019
new/usr/src/uts/common/io/cmlb.c
10570 Need workaround to EFI boot on AMI BIOS
***********************************************************
/*
 * 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 2012 DEY Storage Systems, Inc.  All rights reserved.
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 * Copyright 2016 Toomas Soome <tsoome@me.com>
 * Copyright (c) 2019, Joyent, Inc.
 */

/*
 * This module provides support for labeling operations for target
 * drivers.
 */

#include <sys/scsi/scsi.h>
#include <sys/sunddi.h>
#include <sys/dklabel.h>
#include <sys/dkio.h>
#include <sys/vtoc.h>
#include <sys/dktp/fdisk.h>
#include <sys/vtrace.h>
#include <sys/efi_partition.h>
#include <sys/cmlb.h>
#include <sys/cmlb_impl.h>
#if defined(__i386) || defined(__amd64)
#include <sys/fs/dv_node.h>
#endif
#include <sys/ddi_impldefs.h>

/*
 * Driver minor node structure and data table
 */
struct driver_minor_data {
        char    *name;
        minor_t minor;
        int     type;
};
______unchanged_portion_omitted_

static int
cmlb_validate_efi(efi_gpt_t *labp)
```

```
{
        if (labp->efi_gpt_Signature != EFI_SIGNATURE)
                return (EINVAL);
        /* at least 92 bytes in this version of the spec. */
        /* at least 96 bytes in this version of the spec. */
        if (sizeof (efi_gpt_t) - sizeof (labp->efi_gpt_Reserved2) >
            labp->efi_gpt_HeaderSize)
                return (EINVAL);
        /* this should be 128 bytes */
        if (labp->efi_gpt_SizeOfPartitionEntry != sizeof (efi_gpe_t))
                return (EINVAL);
        return (0);
}
______unchanged_portion_omitted_
```

```
**********************************************************
   10466 Tue Apr 16 05:23:08 2019
new/usr/src/uts/common/sys/efi_partition.h
10570 Need workaround to EFI boot on AMI BIOS
**********************************************************
/*
 * 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) 2002, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc.  All rights reserved.
 * Copyright 2014 Toomas Soome <tsoome@me.com>
 * Copyright (c) 2019, Joyent, Inc.
 */

#ifndef _SYS_EFI_PARTITION_H
#define _SYS_EFI_PARTITION_H

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

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * GUID Partition Table Header
 */

#define EFI_LABEL_SIZE  512
#define LEN_EFI_PAD     (EFI_LABEL_SIZE - \
                            ((5 * sizeof (diskaddr_t)) + \
                            (7 * sizeof (uint_t)) + \
                            (8 * sizeof (char)) + \
                            (1 * (sizeof (struct uuid)))))

#define EFI_SIGNATURE   0x5452415020494645ULL

/*
 * Although the EFI spec is clear that sizeof (efi_gpt_t) is a valid value
 * (512), at least one EFI system (AMI v4.6.4.1) incorrectly expects this to be
 * exactly the size of the structure defined in the spec, that is, 92.
 *
 * As the reserved section is never used, the modified value works fine
 * everywhere else.
 */
#define EFI_HEADER_SIZE (offsetof(efi_gpt_t, efi_gpt_Reserved2))

/* EFI Guid Partition Table Header -- little endian on-disk format */
```

```
typedef struct efi_gpt {
        uint64_t        efi_gpt_Signature;
        uint_t          efi_gpt_Revision;
        uint_t          efi_gpt_HeaderSize;
        uint_t          efi_gpt_HeaderCRC32;
        uint_t          efi_gpt_Reserved1;
        diskaddr_t      efi_gpt_MyLBA;
        diskaddr_t      efi_gpt_AlternateLBA;
        diskaddr_t      efi_gpt_FirstUsableLBA;
        diskaddr_t      efi_gpt_LastUsableLBA;
        struct uuid     efi_gpt_DiskGUID;
        diskaddr_t      efi_gpt_PartitionEntryLBA;
        uint_t          efi_gpt_NumberOfPartitionEntries;
        uint_t          efi_gpt_SizeOfPartitionEntry;
        uint_t          efi_gpt_PartitionEntryArrayCRC32;
        char            efi_gpt_Reserved2[LEN_EFI_PAD];
} efi_gpt_t;
______unchanged_portion_omitted_
```

```
**********************************************************
  208996 Tue Apr 16 05:23:09 2019
new/usr/src/uts/sun4v/io/vds.c
10570 Need workaround to EFI boot on AMI BIOS
**********************************************************
/*
 * 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) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright (c) 2019, Joyent, Inc.
 */

/*
 * Virtual disk server
 */


#include <sys/types.h>
#include <sys/conf.h>
#include <sys/crc32.h>
#include <sys/ddi.h>
#include <sys/dkio.h>
#include <sys/file.h>
#include <sys/fs/hsfs_isospec.h>
#include <sys/mdeg.h>
#include <sys/mhd.h>
#include <sys/modhash.h>
#include <sys/note.h>
#include <sys/pathname.h>
#include <sys/sdt.h>
#include <sys/sunddi.h>
#include <sys/sunldi.h>
#include <sys/sysmacros.h>
#include <sys/vio_common.h>
#include <sys/vio_util.h>
#include <sys/vdsk_mailbox.h>
#include <sys/vdsk_common.h>
#include <sys/vtoc.h>
#include <sys/vfs.h>
#include <sys/stat.h>
#include <sys/scsi/impl/uscsi.h>
#include <sys/ontrap.h>
#include <vm/seg_map.h>

#define ONE_MEGABYTE    (1ULL << 20)
#define ONE_GIGABYTE    (1ULL << 30)
#define ONE_TERABYTE    (1ULL << 40)
```

```

/* Virtual disk server initialization flags */
#define VDS_LDI                 0x01
#define VDS_MDEG                0x02

/* Virtual disk server tunable parameters */
#define VDS_RETRIES             5
#define VDS_LDC_DELAY           1000 /* 1 msecs */
#define VDS_DEV_DELAY           10000000 /* 10 secs */
#define VDS_NCHAINS             32

/* Identification parameters for MD, synthetic dkio(7i) structures, etc. */
#define VDS_NAME                "virtual-disk-server"

#define VD_NAME                 "vd"
#define VD_VOLUME_NAME          "vdisk"
#define VD_ASCIILABEL           "Virtual Disk"

#define VD_CHANNEL_ENDPOINT     "channel-endpoint"
#define VD_ID_PROP              "id"
#define VD_BLOCK_DEVICE_PROP    "vds-block-device"
#define VD_BLOCK_DEVICE_OPTS    "vds-block-device-opts"
#define VD_REG_PROP             "reg"

/* Virtual disk initialization flags */
#define VD_DISK_READY           0x01
#define VD_LOCKING              0x02
#define VD_LDC                  0x04
#define VD_DRING                0x08
#define VD_SID                  0x10
#define VD_SEQ_NUM              0x20
#define VD_SETUP_ERROR          0x40

/* Number of backup labels */
#define VD_DSKIMG_NUM_BACKUP    5

/* Timeout for SCSI I/O */
#define VD_SCSI_RDWR_TIMEOUT    30      /* 30 secs */

/*
 * Default number of threads for the I/O queue. In many cases, we will not
 * receive more than 8 I/O requests at the same time. However there are
 * cases (for example during the OS installation) where we can have a lot
 * more (up to the limit of the DRing size).
 */
#define VD_IOQ_NTHREADS         8

/* Maximum number of logical partitions */
#define VD_MAXPART      (NDKMAP + 1)

/*
 * By Solaris convention, slice/partition 2 represents the entire disk;
 * unfortunately, this convention does not appear to be codified.
 */
#define VD_ENTIRE_DISK_SLICE    2

/* Logical block address for EFI */
#define VD_EFI_LBA_GPT          1       /* LBA of the GPT */
#define VD_EFI_LBA_GPE          2       /* LBA of the GPE */

#define VD_EFI_DEV_SET(dev, vdsk, ioctl)        \
        VDSK_EFI_DEV_SET(dev, vdsk, ioctl,      \
            (vdsk)->vdisk_bsize, (vdsk)->vdisk_size)

/*
 * Flags defining the behavior for flushing asynchronous writes used to
 * performed some write I/O requests.
 *
 * The VD_AWFLUSH_IMMEDIATE enables immediate flushing of asynchronous
 * writes. This ensures that data are committed to the backend when the I/O
 * request reply is sent to the guest domain so this prevents any data to
 * be lost in case a service domain unexpectedly crashes.
 *
 * The flag VD_AWFLUSH_DEFER indicates that flushing is deferred to another
 * thread while the request is immediatly marked as completed. In that case,
 * a guest domain can a receive a reply that its write request is completed
 * while data haven't been flushed to disk yet.
 *
 * Flags VD_AWFLUSH_IMMEDIATE and VD_AWFLUSH_DEFER are mutually exclusive.
 */
#define VD_AWFLUSH_IMMEDIATE    0x01    /* immediate flushing */
#define VD_AWFLUSH_DEFER        0x02    /* defer flushing */
#define VD_AWFLUSH_GROUP        0x04    /* group requests before flushing */

/* Driver types */
typedef enum vd_driver {
        VD_DRIVER_UNKNOWN = 0,  /* driver type unknown  */
        VD_DRIVER_DISK,         /* disk driver */
        VD_DRIVER_VOLUME        /* volume driver */
} vd_driver_t;
```

______unchanged_portion_omitted_

```
/*
 * When a slice, volume or file is exported as a single-slice disk, we want
 * the disk backend (i.e. the slice, volume or file) to be entirely mapped
 * as a slice without the addition of any metadata.
 *
 * So when exporting the disk as an EFI disk, we fake a disk with the following
 * layout: (assuming the block size is 512 bytes)
 *
 *                  flabel        +--- flabel_limit
 *                 <------>       v
 *                 0 1 2  L      34                        34+N      P
 *                 +-+-+--+-------+--------------------------+-------+
 *  virtual disk:  |X|T|EE|XXXXXXX|           slice 0        |RRRRRRR|
 *                 +-+-+--+-------+--------------------------+-------+
 *                    ^ ^         :                          :
 *                    | |         :                          :
 *                GPT-+ +-GPE     :                          :
 *                                +--------------------------+
 *  disk backend:                 |     slice/volume/file    |
 *                                +--------------------------+
 *                                0                          N
 *
 * N is the number of blocks in the slice/volume/file.
 *
 * We simulate a disk with N+M blocks, where M is the number of blocks
 * simluated at the beginning and at the end of the disk (blocks 0-34
 * and 34+N-P).
 *
 * The first 34 blocks (0 to 33) are emulated and can not be changed. Blocks 34
 * to 34+N defines slice 0 and are mapped to the exported backend, and we
 * emulate some blocks at the end of the disk (blocks 34+N to P) as a the EFI
 * reserved partition.
 *
 * - block 0 (X) is unused and return 0
 * - block 1 (T) returns a fake EFI GPT (via DKIOCGETEFI)
 * - blocks 2 to L-1 (E) defines a fake EFI GPE (via DKIOCGETEFI)
 * - blocks L to 33 (X) are unused and return 0
 * - blocks 34 to 34+N are mapped to the exported slice, volume or file
 * - blocks 34+N+1 to P define a fake reserved partition and backup label, it
 *   returns 0
```

```
 *
 * Note: if the backend size is not a multiple of the vdisk block size then
 * the very end of the backend will not map to any block of the virtual disk.
 */
static int
vd_setup_partition_efi(vd_t *vd)
{
        efi_gpt_t *gpt;
        efi_gpe_t *gpe;
        struct uuid uuid = EFI_USR;
        struct uuid efi_reserved = EFI_RESERVED;
        uint32_t crc;
        uint64_t s0_start, s0_end, first_u_lba;
        size_t bsize;

        ASSERT(vd->vdisk_bsize > 0);

        bsize = vd->vdisk_bsize;
        /*
         * The minimum size for the label is 16K (EFI_MIN_ARRAY_SIZE)
         * for GPEs plus one block for the GPT and one for PMBR.
         */
        first_u_lba = (EFI_MIN_ARRAY_SIZE / bsize) + 2;
        vd->flabel_limit = (uint_t)first_u_lba;
        vd->flabel_size = VD_LABEL_EFI_SIZE(bsize);
        vd->flabel = kmem_zalloc(vd->flabel_size, KM_SLEEP);
        gpt = VD_LABEL_EFI_GPT(vd, bsize);
        gpe = VD_LABEL_EFI_GPE(vd, bsize);

        /*
         * Adjust the vdisk_size, we emulate the first few blocks
         * for the disk label.
         */
        vd->vdisk_size += first_u_lba;
        s0_start = first_u_lba;
        s0_end = vd->vdisk_size - 1;

        gpt->efi_gpt_Signature = LE_64(EFI_SIGNATURE);
        gpt->efi_gpt_Revision = LE_32(EFI_VERSION_CURRENT);
        gpt->efi_gpt_HeaderSize = LE_32(EFI_HEADER_SIZE);
        gpt->efi_gpt_HeaderSize = LE_32(sizeof (efi_gpt_t));
        gpt->efi_gpt_FirstUsableLBA = LE_64(first_u_lba);
        gpt->efi_gpt_PartitionEntryLBA = LE_64(2ULL);
        gpt->efi_gpt_SizeOfPartitionEntry = LE_32(sizeof (efi_gpe_t));

        UUID_LE_CONVERT(gpe[0].efi_gpe_PartitionTypeGUID, uuid);
        gpe[0].efi_gpe_StartingLBA = LE_64(s0_start);
        gpe[0].efi_gpe_EndingLBA = LE_64(s0_end);

        if (vd_slice_single_slice) {
                gpt->efi_gpt_NumberOfPartitionEntries = LE_32(1);
        } else {
                /* adjust the number of slices */
                gpt->efi_gpt_NumberOfPartitionEntries = LE_32(VD_MAXPART);
                vd->nslices = V_NUMPAR;

                /* define a fake reserved partition */
                UUID_LE_CONVERT(gpe[VD_MAXPART - 1].efi_gpe_PartitionTypeGUID,
                    efi_reserved);
                gpe[VD_MAXPART - 1].efi_gpe_StartingLBA =
                    LE_64(s0_end + 1);
                gpe[VD_MAXPART - 1].efi_gpe_EndingLBA =
                    LE_64(s0_end + EFI_MIN_RESV_SIZE);

                /* adjust the vdisk_size to include the reserved slice */
                vd->vdisk_size += EFI_MIN_RESV_SIZE;
        }

        gpt->efi_gpt_LastUsableLBA = LE_64(vd->vdisk_size - 1);

        /* adjust the vdisk size for the backup GPT and GPE */
        vd->vdisk_size += (EFI_MIN_ARRAY_SIZE / bsize) + 1;
        gpt->efi_gpt_AlternateLBA = LE_64(vd->vdisk_size - 1);

        CRC32(crc, gpe, sizeof (efi_gpe_t) * VD_MAXPART, -1U, crc32_table);
        gpt->efi_gpt_PartitionEntryArrayCRC32 = LE_32(~crc);

        CRC32(crc, gpt, EFI_HEADER_SIZE, -1U, crc32_table);
        CRC32(crc, gpt, sizeof (efi_gpt_t), -1U, crc32_table);
        gpt->efi_gpt_HeaderCRC32 = LE_32(~crc);

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