

```

*****
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
*****
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) 2002, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright 2015 Nexenta Systems, Inc. All rights reserved.
25  * Copyright 2014 Toomas Soome <tsoome@me.com>
26  * Copyright 2018 OmnicOS Community Edition (OmniOSce) Association.
27  * Copyright 2019 Joyent, Inc.
27  * Copyright (c) 2018, Joyent, Inc.
28 */

30 #include <stdio.h>
31 #include <stdlib.h>
32 #include <errno.h>
33 #include <strings.h>
34 #include <unistd.h>
35 #include <smbios.h>
36 #include <uuid/uuid.h>
37 #include <libintl.h>
38 #include <sys/types.h>
39 #include <sys/dkio.h>
40 #include <sys/vtoc.h>
41 #include <sys/mhd.h>
42 #include <sys/param.h>
43 #include <sys/dktp/fdisk.h>
44 #include <sys/efi_partition.h>
45 #include <sys/byteorder.h>
46 #include <sys/ddi.h>

48 /*
49  * The original conversion array used simple array index, but since
50  * we do need to take account of VTOC tag numbers from other systems,
51  * we need to provide tag values too, or the array will grow too large.
52  *
53  * Still we will fabricate the missing p_tag values.
54  */
55 static struct uuid_to_ptag {
56     struct uuid    uuid;
57     ushort_t      p_tag;
58 } conversion_array[] = {
    unchanged_portion_omitted

```

```

307 static int
308 check_label(int fd, dk_efi_t *dk_ioc)
309 {
310     efi_gpt_t      *efi;
311     uint_t         crc;

313     if (efi_ioctl(fd, DKIOCGETEFI, dk_ioc) == -1) {
314         switch (errno) {
315             case EIO:
316                 return (VT_EIO);
317             default:
318                 return (VT_ERROR);
319         }
320     }
321     efi = dk_ioc->dki_data;
322     if (efi->efi_gpt_Signature != LE_64(EFI_SIGNATURE)) {
323         if (efi_debug)
324             (void) fprintf(stderr,
325                 "Bad EFI signature: 0x%llx != 0x%llx\n",
326                 (long long)efi->efi_gpt_Signature,
327                 (long long)LE_64(EFI_SIGNATURE));
328         return (VT_EINVAL);
329     }

331     /*
332     * check CRC of the header; the size of the header should
333     * never be larger than one block
334     */
335     crc = efi->efi_gpt_HeaderCRC32;
336     efi->efi_gpt_HeaderCRC32 = 0;

338     if (((len_t)LE_32(efi->efi_gpt_HeaderSize) > dk_ioc->dki_length) ||
339         crc != LE_32(efi_crc32((unsigned char *)efi,
340             LE_32(efi->efi_gpt_HeaderSize)))) {
341         if (efi_debug)
342             (void) fprintf(stderr,
343                 "Bad EFI CRC: 0x%x != 0x%x\n",
344                 crc, LE_32(efi_crc32((unsigned char *)efi,
345                     LE_32(efi->efi_gpt_HeaderSize)));
344                 crc,
345                 LE_32(efi_crc32((unsigned char *)efi,
346                     sizeof (struct efi_gpt))));
346         return (VT_EINVAL);
347     }

349     return (0);
350 }
    unchanged_portion_omitted

701 /* writes a "protective" MBR */
702 static int
703 write_pmbd(int fd, struct dk_gpt *vtoc)
704 {
705     dk_efi_t      dk_ioc;
706     struct mboot  mb;
707     uchar_t       *cp;
708     diskaddr_t    size_in_lba;
709     uchar_t       *buf;
710     int            len, slot, active;

712     slot = active = 0;

714     hardware_workarounds(&slot, &active);

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

```

```

718     buf = calloc(len, 1);
719     /*
720     * Preserve any boot code and disk signature if the first block is
721     * already an MBR.
722     */
723     dk_ioc.dki_lba = 0;
724     dk_ioc.dki_length = len;
725     /* LINTED -- always longlong aligned */
726     dk_ioc.dki_data = (efi_gpt_t *)buf;
727     if (efi_ioctl(fd, DKIOCGTEEFI, &dk_ioc) == -1) {
728         (void) memcpy(&mb, buf, sizeof(mb));
729         bzero(&mb, sizeof(mb));
730         mb.signature = LE_16(MBB_MAGIC);
731     } else {
732         (void) memcpy(&mb, buf, sizeof(mb));
733         if (mb.signature != LE_16(MBB_MAGIC)) {
734             bzero(&mb, sizeof(mb));
735             mb.signature = LE_16(MBB_MAGIC);
736         }
737     }
738
739     bzero(&mb.parts, sizeof(mb.parts));
740     cp = (uchar_t *)&mb.parts[slot * sizeof(struct ipart)];
741     /* bootable or not */
742     *cp++ = active ? ACTIVE : NOTACTIVE;
743     /* beginning CHS; same as starting LBA (but one-based) */
744     *cp++ = 0x0;
745     *cp++ = 0x2;
746     *cp++ = 0x0;
747     /* beginning CHS; 0xffffffff if not representable */
748     *cp++ = 0xff;
749     *cp++ = 0xff;
750     *cp++ = 0xff;
751     *cp++ = 0xff;
752     /* OS type */
753     *cp++ = EFI_PMBR;
754     /* ending CHS; 0xffffffff if not representable */
755     *cp++ = 0xff;
756     *cp++ = 0xff;
757     *cp++ = 0xff;
758     /* starting LBA: 1 (little endian format) by EFI definition */
759     *cp++ = 0x01;
760     *cp++ = 0x00;
761     *cp++ = 0x00;
762     *cp++ = 0x00;
763     /* ending LBA: last block on the disk (little endian format) */
764     if (size_in_lba == vtoc->efi_last_lba) {
765         if (size_in_lba < 0xffffffff) {
766             *cp++ = (size_in_lba & 0x000000ff);
767             *cp++ = (size_in_lba & 0x0000ff00) >> 8;
768             *cp++ = (size_in_lba & 0x00ff0000) >> 16;
769             *cp++ = (size_in_lba & 0xff000000) >> 24;
770         } else {
771             *cp++ = 0xff;
772             *cp++ = 0xff;
773             *cp++ = 0xff;
774             *cp++ = 0xff;
775         }
776     } else {
777         (void) memcpy(buf, &mb, sizeof(mb));
778         /* LINTED -- always longlong aligned */
779         dk_ioc.dki_data = (efi_gpt_t *)buf;
780         dk_ioc.dki_lba = 0;
781         dk_ioc.dki_length = len;
782         if (efi_ioctl(fd, DKIOCGTEEFI, &dk_ioc) == -1) {
783             free(buf);

```

```

779         switch (errno) {
780             case EIO:
781                 return (VT_EIO);
782             case EINVAL:
783                 return (VT_EINVAL);
784             default:
785                 return (VT_ERROR);
786         }
787     }
788     free(buf);
789     return (0);
790 }

```

unchanged portion omitted

```

973 /*
974 * write EFI label and backup label
975 */
976 int
977 efi_write(int fd, struct dk_gpt *vtoc)
978 {
979     dk_efi_t          dk_ioc;
980     efi_gpt_t         *efi;
981     efi_gpe_t         *efi_parts;
982     int               i, j;
983     struct dk_cinfo   dki_info;
984     int               nblocks;
985     diskaddr_t        lba_backup_gpt_hdr;
986
987     if (ioctl(fd, DKIOCINFO, (caddr_t)&dki_info) == -1) {
988         if (efi_debug)
989             (void) fprintf(stderr, "DKIOCINFO errno 0x%x\n", errno);
990         switch (errno) {
991             case EIO:
992                 return (VT_EIO);
993             case EINVAL:
994                 return (VT_EINVAL);
995             default:
996                 return (VT_ERROR);
997         }
998     }
999
1000     if (check_input(vtoc))
1001         return (VT_EINVAL);
1002
1003     dk_ioc.dki_lba = 1;
1004     if (NBLOCKS(vtoc->efi_nparts, vtoc->efi_lbasize) < 34) {
1005         dk_ioc.dki_length = EFI_MIN_ARRAY_SIZE + vtoc->efi_lbasize;
1006     } else {
1007         dk_ioc.dki_length = NBLOCKS(vtoc->efi_nparts,
1008             vtoc->efi_lbasize) *
1009             vtoc->efi_lbasize;
1010     }
1011
1012     /*
1013     * the number of blocks occupied by GUID partition entry array
1014     */
1015     nblocks = dk_ioc.dki_length / vtoc->efi_lbasize - 1;
1016
1017     /*
1018     * Backup GPT header is located on the block after GUID
1019     * partition entry array. Here, we calculate the address
1020     * for backup GPT header.
1021     */
1022     lba_backup_gpt_hdr = vtoc->efi_last_u_lba + 1 + nblocks;
1023     if ((dk_ioc.dki_data = calloc(1, dk_ioc.dki_length)) == NULL)

```

```

1024         return (VT_ERROR);
1026     efi = dk_ioc.dki_data;

1028     /* stuff user's input into EFI struct */
1029     efi->efi_gpt_Signature = LE_64(EFI_SIGNATURE);
1030     efi->efi_gpt_Revision = LE_32(vtoc->efi_version); /* 0x02000100 */
1031     efi->efi_gpt_HeaderSize = LE_32(EFI_HEADER_SIZE);
1032     efi->efi_gpt_HeaderSize = LE_32(sizeof (struct efi_gpt));
1033     efi->efi_gpt_Reserved1 = 0;
1034     efi->efi_gpt_MyLBA = LE_64(1ULL);
1035     efi->efi_gpt_AlternateLBA = LE_64(lba_backup_gpt_hdr);
1036     efi->efi_gpt_FirstUsableLBA = LE_64(vtoc->efi_first_u_lba);
1037     efi->efi_gpt_LastUsableLBA = LE_64(vtoc->efi_last_u_lba);
1038     efi->efi_gpt_PartitionEntryLBA = LE_64(2ULL);
1039     efi->efi_gpt_NumberOfPartitionEntries = LE_32(vtoc->efi_nparts);
1040     efi->efi_gpt_SizeOfPartitionEntry = LE_32(sizeof (struct efi_gpe));
1041     UUID_LE_CONVERT(efi->efi_gpt_DiskGUID, vtoc->efi_disk_uguid);

1042     /* LINTED -- always longlong aligned */
1043     efi_parts = (efi_gpe_t *)((char *)dk_ioc.dki_data + vtoc->efi_lbasize);

1045     for (i = 0; i < vtoc->efi_nparts; i++) {
1046         for (j = 0;
1047              j < sizeof (conversion_array) /
1048              sizeof (struct uuid_to_ptag); j++) {
1050             if (vtoc->efi_parts[i].p_tag ==
1051                 conversion_array[j].p_tag) {
1052                 UUID_LE_CONVERT(
1053                     efi_parts[i].efi_gpe_PartitionTypeGUID,
1054                     conversion_array[j].uuid);
1055                 break;
1056             }
1057         }

1059         if (j == sizeof (conversion_array) /
1060             sizeof (struct uuid_to_ptag)) {
1061             /*
1062              * If we didn't have a matching uuid match, bail here.
1063              * Don't write a label with unknown uuid.
1064              */
1065             if (efi_debug) {
1066                 (void) fprintf(stderr,
1067                     "Unknown uuid for p_tag %d\n",
1068                     vtoc->efi_parts[i].p_tag);
1069             }
1070             return (VT_EINVAL);
1071         }

1073         efi_parts[i].efi_gpe_StartingLBA =
1074             LE_64(vtoc->efi_parts[i].p_start);
1075         efi_parts[i].efi_gpe_EndingLBA =
1076             LE_64(vtoc->efi_parts[i].p_start +
1077                 vtoc->efi_parts[i].p_size - 1);
1078         efi_parts[i].efi_gpe_Attributes.PartitionAttrs =
1079             LE_16(vtoc->efi_parts[i].p_flag);
1080         for (j = 0; j < EFI_PART_NAME_LEN; j++) {
1081             efi_parts[i].efi_gpe_PartitionName[j] =
1082                 LE_16((ushort_t)vtoc->efi_parts[i].p_name[j]);
1083         }
1084         if ((vtoc->efi_parts[i].p_tag != V_UNASSIGNED) &&
1085             uuid_is_null((uchar_t *)&vtoc->efi_parts[i].p_uguid)) {
1086             (void) uuid_generate((uchar_t *)
1087                 &vtoc->efi_parts[i].p_uguid);
1088         }

```

```

1089         bcopy(&vtoc->efi_parts[i].p_uguid,
1090             &efi_parts[i].efi_gpe_UniquePartitionGUID,
1091             sizeof (uuid_t));
1092     }
1093     efi->efi_gpt_PartitionEntryArrayCRC32 =
1094         LE_32(efi_crc32((unsigned char *)efi_parts,
1095             vtoc->efi_nparts * (int)sizeof (struct efi_gpe)));
1096     efi->efi_gpt_HeaderCRC32 = LE_32(efi_crc32((unsigned char *)efi,
1097         EFI_HEADER_SIZE));
1098     efi->efi_gpt_HeaderCRC32 =
1099         LE_32(efi_crc32((unsigned char *)efi, sizeof (struct efi_gpt)));

1099     if (efi_ioctl(fd, DKIOCSETEFI, &dk_ioc) == -1) {
1100         free(dk_ioc.dki_data);
1101         switch (errno) {
1102             case EIO:
1103                 return (VT_EIO);
1104             case EINVAL:
1105                 return (VT_EINVAL);
1106             default:
1107                 return (VT_ERROR);
1108         }
1109     }

1111     /* write backup partition array */
1112     dk_ioc.dki_lba = vtoc->efi_last_u_lba + 1;
1113     dk_ioc.dki_length -= vtoc->efi_lbasize;
1114     /* LINTED */
1115     dk_ioc.dki_data = (efi_gpt_t *)((char *)dk_ioc.dki_data +
1116         vtoc->efi_lbasize);

1118     if (efi_ioctl(fd, DKIOCSETEFI, &dk_ioc) == -1) {
1119         /*
1120          * we wrote the primary label okay, so don't fail
1121          */
1122         if (efi_debug) {
1123             (void) fprintf(stderr,
1124                 "write of backup partitions to block %llu "
1125                 "failed, errno %d\n",
1126                 vtoc->efi_last_u_lba + 1,
1127                 errno);
1128         }
1129     }
1130     /*
1131      * now swap MyLBA and AlternateLBA fields and write backup
1132      * partition table header
1133      */
1134     dk_ioc.dki_lba = lba_backup_gpt_hdr;
1135     dk_ioc.dki_length = vtoc->efi_lbasize;
1136     /* LINTED */
1137     dk_ioc.dki_data = (efi_gpt_t *)((char *)dk_ioc.dki_data -
1138         vtoc->efi_lbasize);
1139     efi->efi_gpt_AlternateLBA = LE_64(1ULL);
1140     efi->efi_gpt_MyLBA = LE_64(lba_backup_gpt_hdr);
1141     efi->efi_gpt_PartitionEntryLBA = LE_64(vtoc->efi_last_u_lba + 1);
1142     efi->efi_gpt_HeaderCRC32 = 0;
1143     efi->efi_gpt_HeaderCRC32 =
1144         LE_32(efi_crc32((unsigned char *)dk_ioc.dki_data, EFI_HEADER_SIZE));
1145     efi->efi_gpt_HeaderCRC32 =
1146         LE_32(efi_crc32((unsigned char *)dk_ioc.dki_data,
1147             sizeof (struct efi_gpt)));

1146     if (efi_ioctl(fd, DKIOCSETEFI, &dk_ioc) == -1) {
1147         if (efi_debug) {
1148             (void) fprintf(stderr,
1149                 "write of backup header to block %llu failed, "
1150                 "errno %d\n",

```

new/usr/src/lib/libefi/common/rdwr\_efi.c

7

```
1151             lba_backup_gpt_hdr,  
1152             errno);  
1153         }  
1154     }  
1155     /* write the PMBR */  
1156     (void) write_pmbr(fd, vtoc);  
1157     free(dk_ioc.dki_data);  
1158     return (0);  
1159 }  
_____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
*****
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) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
23 *
24 * Portions Copyright 2010 Robert Milkowski
25 *
26 * Copyright 2017 Nexenta Systems, Inc. All rights reserved.
27 * Copyright (c) 2012, 2017 by Delphix. All rights reserved.
28 * Copyright (c) 2013, Joyent, Inc. All rights reserved.
29 * Copyright (c) 2014 Integros [integros.com]
30 * Copyright (c) 2019, Joyent, Inc.
31 */

33 /*
34  * ZFS volume emulation driver.
35  *
36  * Makes a DMU object look like a volume of arbitrary size, up to 2^64 bytes.
37  * Volumes are accessed through the symbolic links named:
38  *
39  * /dev/zvol/dsk/<pool_name>/<dataset_name>
40  * /dev/zvol/zdsk/<pool_name>/<dataset_name>
41  *
42  * These links are created by the /dev filesystem (sdev_zvolops.c).
43  * Volumes are persistent through reboot. No user command needs to be
44  * run before opening and using a device.
45  */

47 #include <sys/types.h>
48 #include <sys/param.h>
49 #include <sys/errno.h>
50 #include <sys/uio.h>
51 #include <sys/buf.h>
52 #include <sys/modctl.h>
53 #include <sys/open.h>
54 #include <sys/kmem.h>
55 #include <sys/conf.h>
56 #include <sys/cmn_err.h>
57 #include <sys/stat.h>
58 #include <sys/zap.h>
59 #include <sys/spa.h>
60 #include <sys/spa_impl.h>
61 #include <sys/zio.h>

```

```

62 #include <sys/dmu_traverse.h>
63 #include <sys/dnode.h>
64 #include <sys/dsl_dataset.h>
65 #include <sys/dsl_prop.h>
66 #include <sys/dkio.h>
67 #include <sys/efi_partition.h>
68 #include <sys/byteorder.h>
69 #include <sys/pathname.h>
70 #include <sys/ddi.h>
71 #include <sys/sunddi.h>
72 #include <sys/crc32.h>
73 #include <sys/dirent.h>
74 #include <sys/policy.h>
75 #include <sys/fs/zfs.h>
76 #include <sys/zfs_ioctl.h>
77 #include <sys/mkdev.h>
78 #include <sys/zil.h>
79 #include <sys/refcount.h>
80 #include <sys/zfs_znode.h>
81 #include <sys/zfs_rlock.h>
82 #include <sys/vdev_disk.h>
83 #include <sys/vdev_impl.h>
84 #include <sys/vdev_raidz.h>
85 #include <sys/zvol.h>
86 #include <sys/dumphdr.h>
87 #include <sys/zil_impl.h>
88 #include <sys/dbuf.h>
89 #include <sys/dmu_tx.h>
90 #include <sys/zfeature.h>
91 #include <sys/zio_checksum.h>
92 #include <sys/zil_impl.h>
93 #include <sys/dkioc_free_util.h>
94 #include <sys/zfs_rlock.h>

96 #include "zfs_namecheck.h"

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

101 #define ZVOL_DUMPSIZE          "dumpsize"

103 /*
104  * This lock protects the zfsdev_state structure from being modified
105  * while it's being used, e.g. an open that comes in before a create
106  * finishes. It also protects temporary opens of the dataset so that,
107  * e.g., an open doesn't get a spurious EBUSY.
108  */
109 kmutex_t zfsdev_state_lock;
110 static uint32_t zvol_minors;

112 typedef struct zvol_extent {
113     list_node_t    ze_node;
114     dva_t          ze_dva;          /* dva associated with this extent */
115     uint64_t       ze_nblks;      /* number of blocks in extent */
116 } zvol_extent_t;
117 unchanged_portion_omitted

1480 int
1481 zvol_getefi(void *arg, int flag, uint64_t vs, uint8_t bs)
1482 {
1483     struct uuid uuid = EFI_RESERVED;
1484     efi_gpe_t gpe = { 0 };
1485     uint32_t crc;
1486     dk_efi_t efi;
1487     int length;
1488     char *ptr;

```

```

1490     if (ddi_copyin(arg, &efi, sizeof (dk_efi_t), flag))
1491         return (SET_ERROR(EFAULT));
1492     ptr = (char *) (uintptr_t) efi.dki_data_64;
1493     length = efi.dki_length;
1494     /*
1495      * Some clients may attempt to request a PMBR for the
1496      * zvol. Currently this interface will return EINVAL to
1497      * such requests. These requests could be supported by
1498      * adding a check for lba == 0 and consing up an appropriate
1499      * PMBR.
1500      */
1501     if (efi.dki_lba < 1 || efi.dki_lba > 2 || length <= 0)
1502         return (SET_ERROR(EINVAL));
1503
1504     gpe.efi_gpe_StartingLBA = LE_64(34ULL);
1505     gpe.efi_gpe_EndingLBA = LE_64((vs >> bs) - 1);
1506     UUID_LE_CONVERT(gpe.efi_gpe_PartitionTypeGUID, uuid);
1507
1508     if (efi.dki_lba == 1) {
1509         efi_gpt_t gpt = { 0 };
1510
1511         gpt.efi_gpt_Signature = LE_64(EFI_SIGNATURE);
1512         gpt.efi_gpt_Revision = LE_32(EFI_VERSION_CURRENT);
1513         gpt.efi_gpt_HeaderSize = LE_32(EFI_HEADER_SIZE);
1514         gpt.efi_gpt_HeaderSize = LE_32(sizeof (gpt));
1515         gpt.efi_gpt_MyLBA = LE_64(1ULL);
1516         gpt.efi_gpt_FirstUsableLBA = LE_64(34ULL);
1517         gpt.efi_gpt_LastUsableLBA = LE_64((vs >> bs) - 1);
1518         gpt.efi_gpt_PartitionEntryLBA = LE_64(2ULL);
1519         gpt.efi_gpt_NumberOfPartitionEntries = LE_32(1);
1520         gpt.efi_gpt_SizeOfPartitionEntry =
1521             LE_32(sizeof (efi_gpe_t));
1522         CRC32(crc, &gpe, sizeof (gpe), -1U, crc32_table);
1523         gpt.efi_gpt_PartitionEntryArrayCRC32 = LE_32(~crc);
1524         CRC32(crc, &gpt, EFI_HEADER_SIZE, -1U, crc32_table);
1525         CRC32(crc, &gpt, sizeof (gpt), -1U, crc32_table);
1526         gpt.efi_gpt_HeaderCRC32 = LE_32(~crc);
1527         if (ddi_copyout(&gpt, ptr, MIN(sizeof (gpt), length),
1528             flag))
1529             return (SET_ERROR(EFAULT));
1530         ptr += sizeof (gpt);
1531         length -= sizeof (gpt);
1532     }
1533     if (length > 0 && ddi_copyout(&gpe, ptr, MIN(sizeof (gpe),
1534         length), flag))
1535         return (SET_ERROR(EFAULT));
1536     return (0);
1537 }

```

unchanged\_portion\_omitted\_

new/usr/src/uts/common/io/cmlb.c

1

```
*****
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
*****
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 2012 DEY Storage Systems, Inc. All rights reserved.
24  * Copyright 2010 Sun Microsystems, Inc. All rights reserved.
25  * Use is subject to license terms.
26  * Copyright 2016 Toomas Soome <tsoome@me.com>
27  * Copyright (c) 2019, Joyent, Inc.
28 */

30 /*
31  * This module provides support for labeling operations for target
32  * drivers.
33 */

35 #include <sys/scsi/scsi.h>
36 #include <sys/sunddi.h>
37 #include <sys/dklabel.h>
38 #include <sys/dkio.h>
39 #include <sys/vtoc.h>
40 #include <sys/dktp/fdisk.h>
41 #include <sys/vtrace.h>
42 #include <sys/efi_partition.h>
43 #include <sys/cmlb.h>
44 #include <sys/cmlb_impl.h>
45 #if defined(__i386) || defined(__amd64)
46 #include <sys/fs/dv_node.h>
47 #endif
48 #include <sys/ddi_impldefs.h>

50 /*
51  * Driver minor node structure and data table
52  */
53 struct driver_minor_data {
54     char    *name;
55     minor_t minor;
56     int     type;
57 };
_____unchanged_portion_omitted_____

2746 static int
2747 cmlb_validate_efi(efi_gpt_t *labp)
```

new/usr/src/uts/common/io/cmlb.c

2

```
2748 {
2749     if (labp->efi_gpt_Signature != EFI_SIGNATURE)
2750         return (EINVAL);
2751     /* at least 92 bytes in this version of the spec. */
2752     /* at least 96 bytes in this version of the spec. */
2753     if (sizeof (efi_gpt_t) - sizeof (labp->efi_gpt_Reserved2) >
2754         labp->efi_gpt_HeaderSize)
2755         return (EINVAL);
2756     /* this should be 128 bytes */
2757     if (labp->efi_gpt_SizeOfPartitionEntry != sizeof (efi_gpe_t))
2758         return (EINVAL);
2759 }
_____unchanged_portion_omitted_____
```

new/usr/src/uts/common/sys/efi\_partition.h

1

```
*****
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
*****
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) 2002, 2010, Oracle and/or its affiliates. All rights reserved.
23 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
24 * Copyright 2014 Toomas Soome <tsoome@me.com>
25 * Copyright (c) 2019, Joyent, Inc.
26 */

28 #ifndef _SYS_EFI_PARTITION_H
29 #define _SYS_EFI_PARTITION_H

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

34 #ifdef __cplusplus
35 extern "C" {
36 #endif

38 /*
39  * GUID Partition Table Header
40  */

42 #define EFI_LABEL_SIZE 512
43 #define LEN_EFI_PAD (EFI_LABEL_SIZE - \
44 (5 * sizeof (diskaddr_t)) + \
45 (7 * sizeof (uint_t)) + \
46 (8 * sizeof (char)) + \
47 (1 * (sizeof (struct uuid))))

49 #define EFI_SIGNATURE 0x5452415020494645ULL

51 /*
52 * Although the EFI spec is clear that sizeof (efi_gpt_t) is a valid value
53 * (512), at least one EFI system (AMI v4.6.4.1) incorrectly expects this to be
54 * exactly the size of the structure defined in the spec, that is, 92.
55 *
56 * As the reserved section is never used, the modified value works fine
57 * everywhere else.
58 */
59 #define EFI_HEADER_SIZE (offsetof(efi_gpt_t, efi_gpt_Reserved2))

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

new/usr/src/uts/common/sys/efi\_partition.h

2

```
62 typedef struct efi_gpt {
63     uint64_t efi_gpt_Signature;
64     uint_t efi_gpt_Revision;
65     uint_t efi_gpt_HeaderSize;
66     uint_t efi_gpt_HeaderCRC32;
67     uint_t efi_gpt_Reserved1;
68     diskaddr_t efi_gpt_MyLBA;
69     diskaddr_t efi_gpt_AlternateLBA;
70     diskaddr_t efi_gpt_FirstUsableLBA;
71     diskaddr_t efi_gpt_LastUsableLBA;
72     struct uuid efi_gpt_DiskGUID;
73     diskaddr_t efi_gpt_PartitionEntryLBA;
74     uint_t efi_gpt_NumberOfPartitionEntries;
75     uint_t efi_gpt_SizeOfPartitionEntry;
76     uint_t efi_gpt_PartitionEntryArrayCRC32;
77     char efi_gpt_Reserved2[LEN_EFI_PAD];
78 } 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
*****
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) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
24  * Copyright (c) 2019, Joyent, Inc.
25  */

27 /*
28  * Virtual disk server
29  */

32 #include <sys/types.h>
33 #include <sys/conf.h>
34 #include <sys/crc32.h>
35 #include <sys/ddi.h>
36 #include <sys/dkio.h>
37 #include <sys/file.h>
38 #include <sys/fs/hsfs_isospec.h>
39 #include <sys/mdeg.h>
40 #include <sys/mhd.h>
41 #include <sys/modhash.h>
42 #include <sys/note.h>
43 #include <sys/pathname.h>
44 #include <sys/sdt.h>
45 #include <sys/sunddi.h>
46 #include <sys/sunldi.h>
47 #include <sys/sysmacros.h>
48 #include <sys/vio_common.h>
49 #include <sys/vio_util.h>
50 #include <sys/vdsk_mailbox.h>
51 #include <sys/vdsk_common.h>
52 #include <sys/vtoc.h>
53 #include <sys/vfs.h>
54 #include <sys/stat.h>
55 #include <sys/scsi/impl/uscsi.h>
56 #include <sys/onttrap.h>
57 #include <vm/seg_map.h>

59 #define ONE_MEGABYTE    (1ULL << 20)
60 #define ONE_GIGABYTE    (1ULL << 30)
61 #define ONE_TERABYTE    (1ULL << 40)

```

```

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

67 /* Virtual disk server tunable parameters */
68 #define VDS_RETRIES     5
69 #define VDS_LDC_DELAY   1000 /* 1 msecs */
70 #define VDS_DEV_DELAY   1000000 /* 10 secs */
71 #define VDS_NCHAINS     32

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

76 #define VD_NAME         "vd"
77 #define VD_VOLUME_NAME "vdisk"
78 #define VD_ASCIIDLABEL "Virtual Disk"

80 #define VD_CHANNEL_ENDPOINT "channel-endpoint"
81 #define VD_ID_PROP        "id"
82 #define VD_BLOCK_DEVICE_PROP "vds-block-device"
83 #define VD_BLOCK_DEVICE_OPTS "vds-block-device-opts"
84 #define VD_REG_PROP      "reg"

86 /* Virtual disk initialization flags */
87 #define VD_DISK_READY    0x01
88 #define VD_LOCKING      0x02
89 #define VD_LDC          0x04
90 #define VD_DRING        0x08
91 #define VD_SID          0x10
92 #define VD_SEQ_NUM      0x20
93 #define VD_SETUP_ERROR  0x40

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

98 /* Timeout for SCSI I/O */
99 #define VD SCSI_RDWR_TIMEOUT 30 /* 30 secs */

101 /*
102  * Default number of threads for the I/O queue. In many cases, we will not
103  * receive more than 8 I/O requests at the same time. However there are
104  * cases (for example during the OS installation) where we can have a lot
105  * more (up to the limit of the DRing size).
106  */
107 #define VD_IOQ_NTHREADS 8

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

112 /*
113  * By Solaris convention, slice/partition 2 represents the entire disk;
114  * unfortunately, this convention does not appear to be codified.
115  */
116 #define VD_ENTIRE_DISK_SLICE 2

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

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

126 /*
127  * Flags defining the behavior for flushing asynchronous writes used to

```

```

128 * performed some write I/O requests.
129 *
130 * The VD_AWFLUSH_IMMEDIATE enables immediate flushing of asynchronous
131 * writes. This ensures that data are committed to the backend when the I/O
132 * request reply is sent to the guest domain so this prevents any data to
133 * be lost in case a service domain unexpectedly crashes.
134 *
135 * The flag VD_AWFLUSH_DEFER indicates that flushing is deferred to another
136 * thread while the request is immediately marked as completed. In that case,
137 * a guest domain can receive a reply that its write request is completed
138 * while data haven't been flushed to disk yet.
139 *
140 * Flags VD_AWFLUSH_IMMEDIATE and VD_AWFLUSH_DEFER are mutually exclusive.
141 */
142 #define VD_AWFLUSH_IMMEDIATE 0x01 /* immediate flushing */
143 #define VD_AWFLUSH_DEFER 0x02 /* defer flushing */
144 #define VD_AWFLUSH_GROUP 0x04 /* group requests before flushing */

146 /* Driver types */
147 typedef enum vd_driver {
148     VD_DRIVER_UNKNOWN = 0, /* driver type unknown */
149     VD_DRIVER_DISK, /* disk driver */
150     VD_DRIVER_VOLUME /* volume driver */
151 } vd_driver_t;
unchanged portion omitted

5757 /*
5758 * When a slice, volume or file is exported as a single-slice disk, we want
5759 * the disk backend (i.e. the slice, volume or file) to be entirely mapped
5760 * as a slice without the addition of any metadata.
5761 *
5762 * So when exporting the disk as an EFI disk, we fake a disk with the following
5763 * layout: (assuming the block size is 512 bytes)
5764 *
5765 *          flabel          +--- flabel_limit
5766 *          <----->      v
5767 *          0 1 2 L 34                                34+N    P
5768 *          +-----+-----+-----+-----+-----+
5769 * virtual disk: |X|T|EE|XXXXXXX|                slice 0    |RRRRRRR|
5770 *          +-----+-----+-----+-----+-----+
5771 *          ^         ^         ^         ^         ^
5772 *          |         |         |         |         |
5773 *          GPT-+ +-GPE :         :         :         :
5774 *          +-----+-----+-----+-----+
5775 * disk backend: |                slice/volume/file |
5776 *          +-----+-----+-----+-----+
5777 *          0                                N
5778 *
5779 * N is the number of blocks in the slice/volume/file.
5780 *
5781 * We simulate a disk with N+M blocks, where M is the number of blocks
5782 * simulated at the beginning and at the end of the disk (blocks 0-34
5783 * and 34+N-P).
5784 *
5785 * The first 34 blocks (0 to 33) are emulated and can not be changed. Blocks 34
5786 * to 34+N defines slice 0 and are mapped to the exported backend, and we
5787 * emulate some blocks at the end of the disk (blocks 34+N to P) as a the EFI
5788 * reserved partition.
5789 *
5790 * - block 0 (X) is unused and return 0
5791 * - block 1 (T) returns a fake EFI GPT (via DKIOCGETEFI)
5792 * - blocks 2 to L-1 (E) defines a fake EFI GPE (via DKIOCGETEFI)
5793 * - blocks L to 33 (X) are unused and return 0
5794 * - blocks 34 to 34+N are mapped to the exported slice, volume or file
5795 * - blocks 34+N+1 to P define a fake reserved partition and backup label, it
5796 * returns 0

```

```

5797 *
5798 * Note: if the backend size is not a multiple of the vdisk block size then
5799 * the very end of the backend will not map to any block of the virtual disk.
5800 */
5801 static int
5802 vd_setup_partition_efi(vd_t *vd)
5803 {
5804     efi_gpt_t *gpt;
5805     efi_gpe_t *gpe;
5806     struct uuid uuid = EFI_USR;
5807     struct uuid efi_reserved = EFI_RESERVED;
5808     uint32_t crc;
5809     uint64_t s0_start, s0_end, first_u_lba;
5810     size_t bsize;

5812     ASSERT(vd->vdisk_bsize > 0);

5814     bsize = vd->vdisk_bsize;
5815     /*
5816      * The minimum size for the label is 16K (EFI_MIN_ARRAY_SIZE)
5817      * for GPEs plus one block for the GPT and one for PMBR.
5818      */
5819     first_u_lba = (EFI_MIN_ARRAY_SIZE / bsize) + 2;
5820     vd->flabel_limit = (uint_t)first_u_lba;
5821     vd->flabel_size = VD_LABEL_EFI_SIZE(bsize);
5822     vd->flabel = kmem_zalloc(vd->flabel_size, KM_SLEEP);
5823     gpt = VD_LABEL_EFI_GPT(vd, bsize);
5824     gpe = VD_LABEL_EFI_GPE(vd, bsize);

5826     /*
5827      * Adjust the vdisk_size, we emulate the first few blocks
5828      * for the disk label.
5829      */
5830     vd->vdisk_size += first_u_lba;
5831     s0_start = first_u_lba;
5832     s0_end = vd->vdisk_size - 1;

5834     gpt->efi_gpt_Signature = LE_64(EFI_SIGNATURE);
5835     gpt->efi_gpt_Revision = LE_32(EFI_VERSION_CURRENT);
5836     gpt->efi_gpt_HeaderSize = LE_32(EFI_HEADER_SIZE);
5837     gpt->efi_gpt_HeaderSize = LE_32(sizeof(efi_gpt_t));
5838     gpt->efi_gpt_FirstUsableLBA = LE_64(first_u_lba);
5839     gpt->efi_gpt_PartitionEntryLBA = LE_64(2ULL);
5840     gpt->efi_gpt_SizeOfPartitionEntry = LE_32(sizeof(efi_gpe_t));

5841     UUID_LE_CONVERT(gpe[0].efi_gpe_PartitionTypeGUID, uuid);
5842     gpe[0].efi_gpe_StartingLBA = LE_64(s0_start);
5843     gpe[0].efi_gpe_EndingLBA = LE_64(s0_end);

5845     if (vd_slice_single_slice) {
5846         gpt->efi_gpt_NumberOfPartitionEntries = LE_32(1);
5847     } else {
5848         /* adjust the number of slices */
5849         gpt->efi_gpt_NumberOfPartitionEntries = LE_32(VD_MAXPART);
5850         vd->nslices = V_NUMPAR;

5852         /* define a fake reserved partition */
5853         UUID_LE_CONVERT(gpe[VD_MAXPART - 1].efi_gpe_PartitionTypeGUID,
5854             efi_reserved);
5855         gpe[VD_MAXPART - 1].efi_gpe_StartingLBA =
5856             LE_64(s0_end + 1);
5857         gpe[VD_MAXPART - 1].efi_gpe_EndingLBA =
5858             LE_64(s0_end + EFI_MIN_RESV_SIZE);

5860         /* adjust the vdisk_size to include the reserved slice */
5861         vd->vdisk_size += EFI_MIN_RESV_SIZE;

```

```
5862     }
5864     gpt->efi_gpt_LastUsableLBA = LE_64(vd->vdisk_size - 1);
5866     /* adjust the vdisk size for the backup GPT and GPE */
5867     vd->vdisk_size += (EFI_MIN_ARRAY_SIZE / bsize) + 1;
5868     gpt->efi_gpt_AlternateLBA = LE_64(vd->vdisk_size - 1);
5870     CRC32(crc, gpe, sizeof (efi_gpe_t) * VD_MAXPART, -1U, crc32_table);
5871     gpt->efi_gpt_PartitionEntryArrayCRC32 = LE_32(~crc);
5873     CRC32(crc, gpt, EFI_HEADER_SIZE, -1U, crc32_table);
5872     CRC32(crc, gpt, sizeof (efi_gpt_t), -1U, crc32_table);
5874     gpt->efi_gpt_HeaderCRC32 = LE_32(~crc);
5876     return (0);
5877 }
_____unchanged_portion_omitted_____
```