****
14245 Thu Jan 17 14:25:03 2019
new/usr/src/cmd/checknr/checknr.c
10100 Illumos is confused about calloc() arguments
****

```
   1 /*      Copyright (c) 1984, 1986, 1987, 1988, 1989 AT&T */
   2 /*        All Rights Reserved   */


   5 /*
   6  * Copyright (c) 1980 Regents of the University of California.
   7  * All rights reserved. The Berkeley software License Agreement
   8  * specifies the terms and conditions for redistribution.
   9  */

  11 /*
  12  * Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
  13  * Use is subject to license terms.
  14  */

  16 /*
  17  * Copyright (c) 2018, Joyent, Inc.
  18  */
  16 #pragma ident   "%Z%%M% %I%     %E% SMI"

  20 /*
  21  * checknr: check an nroff/troff input file for matching macro calls.
  22  * we also attempt to match size and font changes, but only the embedded
  23  * kind.  These must end in \s0 and \fP resp.  Maybe more sophistication
  24  * later but for now think of these restrictions as contributions to
  25  * structured typesetting.
  26  */
  27 #include <stdio.h>
  28 #include <stdlib.h>
  29 #include <unistd.h>
  30 #include <string.h>
  31 #include <ctype.h>
  32 #include <locale.h>

  34 #define MAXSTK  100     /* Stack size */
  35 static  int     maxstk;
  36 #define MAXBR   100     /* Max number of bracket pairs known */
  37 #define MAXCMDS 500     /* Max number of commands known */

  39 /*
  40  * The stack on which we remember what we've seen so far.
  41  */
  42 static struct stkstr {
  43         int opno;       /* number of opening bracket */
  44         int pl;         /* '+', '-', ' ' for \s, 1 for \f, 0 for .ft */
  45         int parm;       /* parm to size, font, etc */
  46         int lno;        /* line number the thing came in in */
  47 } *stk;
```

*_____unchanged_portion_omitted_*

```
 180 int
 181 main(argc, argv)
 182 int argc;
 183 char **argv;
 184 {
 185         FILE *f;
 186         int i;
 187         char *cp;
 188         char b1[4];

 190         (void) setlocale(LC_ALL, "");
```



```
 191 #if !defined(TEXT_DOMAIN)
 192 #define TEXT_DOMAIN     "SYS_TEST"
 193 #endif
 194         (void) textdomain(TEXT_DOMAIN);
 195         stk = (struct stkstr *)calloc(100, sizeof (struct stkstr));
 193         stk = (struct stkstr *)calloc(sizeof (struct stkstr), 100);
 196         maxstk = 100;
 197         /* Figure out how many known commands there are */
 198         while (knowncmds[ncmds])
 199                 ncmds++;
 200         while (argc > 1 && argv[1][0] == '-') {
 201                 switch (argv[1][1]) {

 203                 /* -a: add pairs of macros */
 204                 case 'a':
 205                         i = strlen(argv[1]) - 2;
 206                         if (i % 6 != 0)
 207                                 usage();
 208                         /* look for empty macro slots */
 209                         for (i = 0; br[i].opbr; i++)
 210                                 ;
 211                         for (cp = argv[1]+3; cp[-1]; cp += 6) {
 212                                 br[i].opbr = malloc(3);
 213                                 (void) strncpy(br[i].opbr, cp, 2);
 214                                 br[i].clbr = malloc(3);
 215                                 (void) strncpy(br[i].clbr, cp+3, 2);
 216                                 /* knows pairs are also known cmds */
 217                                 addmac(br[i].opbr);
 218                                 addmac(br[i].clbr);
 219                                 i++;
 220                         }
 221                         break;

 223                 /* -c: add known commands */
 224                 case 'c':
 225                         i = strlen(argv[1]) - 2;
 226                         if (i % 3 != 0)
 227                                 usage();
 228                         for (cp = argv[1]+3; cp[-1]; cp += 3) {
 229                                 if (cp[2] && cp[2] != '.')
 230                                         usage();
 231                                 (void) strncpy(b1, cp, 2);
 232                                 addmac(b1);
 233                         }
 234                         break;

 236                 /* -f: ignore font changes */
 237                 case 'f':
 238                         fflag = 1;
 239                         break;

 241                 /* -s: ignore size changes */
 242                 case 's':
 243                         sflag = 1;
 244                         break;
 245                 default:
 246                         usage();
 247                 }
 248                 argc--; argv++;
 249         }

 251         nfiles = argc - 1;

 253         if (nfiles > 0) {
 254                 for (i = 1; i < argc; i++) {
 255                         cfilename = argv[i];
```

```
256                         f = fopen(cfilename, "r");
257                         if (f == NULL) {
258                                 perror(cfilename);
259                                 exit(1);
260                                 }
261                         else
262                                 process(f);
263                 }
264         } else {
265                 cfilename = "stdin";
266                 process(stdin);
267         }
268         return (0);
269 }
```

unchanged_portion_omitted_

```
**********************************************************
    7733 Thu Jan 17 14:25:04 2019
new/usr/src/cmd/sgs/librtld_db/common/rtld_db.c
10100 Illumos is confused about calloc() arguments
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 2018, Joyent, Inc.
 */
#pragma ident   "%Z%%M% %I%     %E% SMI"


#include        <stdlib.h>
#include        <stdio.h>
#include        <string.h>
#include        <proc_service.h>
#include        <link.h>
#include        <rtld_db.h>
#include        <rtld.h>
#include        <_rtld_db.h>
#include        <msg.h>
#include        <sys/param.h>

/*
 * Mutex to protect global data
 */
mutex_t glob_mutex = DEFAULTMUTEX;
int     rtld_db_version = RD_VERSION1;
int     rtld_db_logging = 0;
char    rtld_db_helper_path[MAXPATHLEN];


void
rd_log(const int on_off)
{
        (void) mutex_lock(&glob_mutex);
        rtld_db_logging = on_off;
        (void) mutex_unlock(&glob_mutex);
        LOG(ps_plog(MSG_ORIG(MSG_DB_LOGENABLE)));
}
```

_____unchanged_portion_omitted_


```
rd_agent_t *
rd_new(struct ps_prochandle *php)
{
        rd_agent_t      *rap;

        LOG(ps_plog(MSG_ORIG(MSG_DB_RDNEW), php));
        if ((rap = (rd_agent_t *)calloc(1, sizeof (rd_agent_t))) == NULL)
        if ((rap = (rd_agent_t *)calloc(sizeof (rd_agent_t), 1)) == NULL)
                return (0);

        rap->rd_psp = php;
        (void) mutex_init(&rap->rd_mutex, USYNC_THREAD, 0);
        if (rd_reset(rap) != RD_OK) {
                if (rap->rd_helper.rh_dlhandle != NULL) {
                        rap->rd_helper.rh_ops->rho_fini(rap->rd_helper.rh_data);
                        (void) dlclose(rap->rd_helper.rh_dlhandle);
                }
                free(rap);
                LOG(ps_plog(MSG_ORIG(MSG_DB_RESETFAIL)));
                return ((rd_agent_t *)0);
        }

        return (rap);
}
```

_____unchanged_portion_omitted_

****
35482 Thu Jan 17 14:25:04 2019
new/usr/src/lib/libefi/common/rdwr_efi.c
10100 Illumos is confused about calloc() arguments
****

```
/*
 * 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 (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_*

```
/*
```



```
 * the number of blocks the EFI label takes up (round up to nearest
 * block)
 */
#define NBLOCKS(p, l)   (1 + ((((p) * (int)sizeof (efi_gpe_t))  + \
                                ((l) - 1)) / (l)))
/* number of partitions -- limited by what we can malloc */
#define MAX_PARTS       ((4294967295UL - sizeof (struct dk_gpt)) / \
                            sizeof (struct dk_part))

int
efi_alloc_and_init(int fd, uint32_t nparts, struct dk_gpt **vtoc)
{
        diskaddr_t      capacity;
        uint_t          lbsize;
        uint_t          nblocks;
        size_t          length;
        struct dk_gpt   *vptr;
        struct uuid     uuid;

        if (read_disk_info(fd, &capacity, &lbsize) != 0) {
                if (efi_debug)
                        (void) fprintf(stderr,
                            "couldn't read disk information\n");
                return (-1);
        }

        nblocks = NBLOCKS(nparts, lbsize);
        if ((nblocks * lbsize) < EFI_MIN_ARRAY_SIZE + lbsize) {
                /* 16K plus one block for the GPT */
                nblocks = EFI_MIN_ARRAY_SIZE / lbsize + 1;
        }

        if (nparts > MAX_PARTS) {
                if (efi_debug) {
                        (void) fprintf(stderr,
                        "the maximum number of partitions supported is %lu\n",
                            MAX_PARTS);
                }
                return (-1);
        }

        length = sizeof (struct dk_gpt) +
            sizeof (struct dk_part) * (nparts - 1);

        if ((*vtoc = calloc(1, length)) == NULL)
        if ((*vtoc = calloc(length, 1)) == NULL)
                return (-1);

        vptr = *vtoc;

        vptr->efi_version = EFI_VERSION_CURRENT;
        vptr->efi_lbasize = lbsize;
        vptr->efi_nparts = nparts;
        /*
         * add one block here for the PMBR; on disks with a 512 byte
         * block size and 128 or fewer partitions, efi_first_u_lba
         * should work out to "34"
         */
        vptr->efi_first_u_lba = nblocks + 1;
        vptr->efi_last_lba = capacity - 1;
        vptr->efi_altern_lba = capacity -1;
        vptr->efi_last_u_lba = vptr->efi_last_lba - nblocks;

        (void) uuid_generate((uchar_t *)&uuid);
        UUID_LE_CONVERT(vptr->efi_disk_uguid, uuid);
        return (0);
```

```
213 }

215 /*
216  * Read EFI - return partition number upon success.
217  */
218 int
219 efi_alloc_and_read(int fd, struct dk_gpt **vtoc)
220 {
221         int                     rval;
222         uint32_t                nparts;
223         int                     length;
224         struct mboot            *mbr;
225         struct ipart            *ipart;
226         diskaddr_t              capacity;
227         uint_t                  lbsize;
228         int                     i;

230         if (read_disk_info(fd, &capacity, &lbsize) != 0)
231                 return (VT_ERROR);

233         if ((mbr = calloc(1, lbsize)) == NULL)
232         (( (mbr = calloc(lbsize, 1)) == NULL)
234                 return (VT_ERROR);

236         if ((ioctl(fd, DKIOCGMBOOT, (caddr_t)mbr)) == -1) {
237                 free(mbr);
238                 return (VT_ERROR);
239         }

241         if (mbr->signature != MBB_MAGIC) {
242                 free(mbr);
243                 return (VT_EINVAL);
244         }
245         ipart = (struct ipart *)(uintptr_t)mbr->parts;

247         /* Check if we have partition with ID EFI_PMBR */
248         for (i = 0; i < FD_NUMPART; i++) {
249                 if (ipart[i].systid == EFI_PMBR)
250                         break;
251         }
252         free(mbr);
253         if (i == FD_NUMPART)
254                 return (VT_EINVAL);

256         /* figure out the number of entries that would fit into 16K */
257         nparts = EFI_MIN_ARRAY_SIZE / sizeof (efi_gpe_t);
258         length = (int) sizeof (struct dk_gpt) +
259             (int) sizeof (struct dk_part) * (nparts - 1);
260         if ((*vtoc = calloc(1, length)) == NULL)
259         (( (*vtoc = calloc(length, 1)) == NULL)
261                 return (VT_ERROR);

263         (*vtoc)->efi_nparts = nparts;
264         rval = efi_read(fd, *vtoc);

266         if ((rval == VT_EINVAL) && (*vtoc)->efi_nparts > nparts) {
267                 void *tmp;
268                 length = (int) sizeof (struct dk_gpt) +
269                     (int) sizeof (struct dk_part) *
270                     ((*vtoc)->efi_nparts - 1);
271                 nparts = (*vtoc)->efi_nparts;
272                 if ((tmp = realloc(*vtoc, length)) == NULL) {
273                         free (*vtoc);
274                         *vtoc = NULL;
275                         return (VT_ERROR);
276                 } else {
```



```
277                         *vtoc = tmp;
278                         rval = efi_read(fd, *vtoc);
279                 }
280         }

282         if (rval < 0) {
283                 if (efi_debug) {
284                         (void) fprintf(stderr,
285                             "read of EFI table failed, rval=%d\n", rval);
286                 }
287                 free (*vtoc);
288                 *vtoc = NULL;
289         }

291         return (rval);
292 }
```

_____***unchanged_portion_omitted_***

```
353 static int
354 efi_read(int fd, struct dk_gpt *vtoc)
355 {
356         int                     i, j;
357         int                     label_len;
358         int                     rval = 0;
359         int                     vdc_flag = 0;
360         struct dk_minfo         disk_info;
361         dk_efi_t                dk_ioc;
362         efi_gpt_t               *efi;
363         efi_gpe_t               *efi_parts;
364         struct dk_cinfo         dki_info;
365         uint32_t                user_length;
366         boolean_t               legacy_label = B_FALSE;

368         /*
369          * get the partition number for this file descriptor.
370          */
371         if (ioctl(fd, DKIOCINFO, (caddr_t)&dki_info) == -1) {
372                 if (efi_debug) {
373                         (void) fprintf(stderr, "DKIOCINFO errno 0x%x\n", errno);
374                 }
375                 switch (errno) {
376                 case EIO:
377                         return (VT_EIO);
378                 case EINVAL:
379                         return (VT_EINVAL);
380                 default:
381                         return (VT_ERROR);
382                 }
383         }

385         if ((strncmp(dki_info.dki_cname, "vdc", 4) == 0) &&
386             (strncmp(dki_info.dki_dname, "vdc", 4) == 0)) {
387                 /*
388                  * The controller and drive name "vdc" (virtual disk client)
389                  * indicates a LDoms virtual disk.
390                  */
391                 vdc_flag++;
392         }

394         /* get the LBA size */
395         if (ioctl(fd, DKIOCGMEDIAINFO, (caddr_t)&disk_info) == -1) {
396                 if (efi_debug) {
397                         (void) fprintf(stderr,
398                             "assuming LBA 512 bytes %d\n",
399                             errno);
400                 }
```

```
401                 disk_info.dki_lbsize = DEV_BSIZE;
402         }
403         if (disk_info.dki_lbsize == 0) {
404                 if (efi_debug) {
405                         (void) fprintf(stderr,
406                             "efi_read: assuming LBA 512 bytes\n");
407                 }
408                 disk_info.dki_lbsize = DEV_BSIZE;
409         }
410         /*
411          * Read the EFI GPT to figure out how many partitions we need
412          * to deal with.
413          */
414         dk_ioc.dki_lba = 1;
415         if (NBLOCKS(vtoc->efi_nparts, disk_info.dki_lbsize) < 34) {
416                 label_len = EFI_MIN_ARRAY_SIZE + disk_info.dki_lbsize;
417         } else {
418                 label_len = vtoc->efi_nparts * (int) sizeof (efi_gpe_t) +
419                     disk_info.dki_lbsize;
420                 if (label_len % disk_info.dki_lbsize) {
421                         /* pad to physical sector size */
422                         label_len += disk_info.dki_lbsize;
423                         label_len &= ~(disk_info.dki_lbsize - 1);
424                 }
425         }

427         if ((dk_ioc.dki_data = calloc(1, label_len)) == NULL)
426         if ((dk_ioc.dki_data = calloc(label_len, 1)) == NULL)
428                 return (VT_ERROR);

430         dk_ioc.dki_length = disk_info.dki_lbsize;
431         user_length = vtoc->efi_nparts;
432         efi = dk_ioc.dki_data;
433         if ((rval = check_label(fd, &dk_ioc)) == VT_EINVAL) {
434                 /*
435                  * No valid label here; try the alternate. Note that here
436                  * we just read GPT header and save it into dk_ioc.data,
437                  * Later, we will read GUID partition entry array if we
438                  * can get valid GPT header.
439                  */

441                 /*
442                  * This is a workaround for legacy systems. In the past, the
443                  * last sector of SCSI disk was invisible on x86 platform. At
444                  * that time, backup label was saved on the next to the last
445                  * sector. It is possible for users to move a disk from previous
446                  * solaris system to present system. Here, we attempt to search
447                  * legacy backup EFI label first.
448                  */
449                 dk_ioc.dki_lba = disk_info.dki_capacity - 2;
450                 dk_ioc.dki_length = disk_info.dki_lbsize;
451                 rval = check_label(fd, &dk_ioc);
452                 if (rval == VT_EINVAL) {
453                         /*
454                          * we didn't find legacy backup EFI label, try to
455                          * search backup EFI label in the last block.
456                          */
457                         dk_ioc.dki_lba = disk_info.dki_capacity - 1;
458                         dk_ioc.dki_length = disk_info.dki_lbsize;
459                         rval = check_label(fd, &dk_ioc);
460                         if (rval == 0) {
461                                 legacy_label = B_TRUE;
462                                 if (efi_debug)
463                                         (void) fprintf(stderr,
464                                             "efi_read: primary label corrupt; "
465                                             "using EFI backup label located on"
```



```
466                                             " the last block\n");
467                         }
468                 } else {
469                         if ((efi_debug) && (rval == 0))
470                                 (void) fprintf(stderr, "efi_read: primary label"
471                                     " corrupt; using legacy EFI backup label "
472                                     " located on the next to last block\n");
473                 }

475                 if (rval == 0) {
476                         dk_ioc.dki_lba = LE_64(efi->efi_gpt_PartitionEntryLBA);
477                         vtoc->efi_flags |= EFI_GPT_PRIMARY_CORRUPT;
478                         vtoc->efi_nparts =
479                             LE_32(efi->efi_gpt_NumberOfPartitionEntries);
480                         /*
481                          * Partition tables are between backup GPT header
482                          * table and ParitionEntryLBA (the starting LBA of
483                          * the GUID partition entries array). Now that we
484                          * already got valid GPT header and saved it in
485                          * dk_ioc.dki_data, we try to get GUID partition
486                          * entry array here.
487                          */
488                         /* LINTED */
489                         dk_ioc.dki_data = (efi_gpt_t *)((char *)dk_ioc.dki_data
490                             + disk_info.dki_lbsize);
491                         if (legacy_label)
492                                 dk_ioc.dki_length = disk_info.dki_capacity - 1 -
493                                     dk_ioc.dki_lba;
494                         else
495                                 dk_ioc.dki_length = disk_info.dki_capacity - 2 -
496                                     dk_ioc.dki_lba;
497                         dk_ioc.dki_length *= disk_info.dki_lbsize;
498                         if (dk_ioc.dki_length >
499                             ((len_t)label_len - sizeof (*dk_ioc.dki_data))) {
500                                 rval = VT_EINVAL;
501                         } else {
502                                 /*
503                                  * read GUID partition entry array
504                                  */
505                                 rval = efi_ioctl(fd, DKIOCGETEFI, &dk_ioc);
506                         }
507                 }

509         } else if (rval == 0) {

511                 dk_ioc.dki_lba = LE_64(efi->efi_gpt_PartitionEntryLBA);
512                 /* LINTED */
513                 dk_ioc.dki_data = (efi_gpt_t *)((char *)dk_ioc.dki_data
514                     + disk_info.dki_lbsize);
515                 dk_ioc.dki_length = label_len - disk_info.dki_lbsize;
516                 rval = efi_ioctl(fd, DKIOCGETEFI, &dk_ioc);

518         } else if (vdc_flag && rval == VT_ERROR && errno == EINVAL) {
519                 /*
520                  * When the device is a LDoms virtual disk, the DKIOCGETEFI
521                  * ioctl can fail with EINVAL if the virtual disk backend
522                  * is a ZFS volume serviced by a domain running an old version
523                  * of Solaris. This is because the DKIOCGETEFI ioctl was
524                  * initially incorrectly implemented for a ZFS volume and it
525                  * expected the GPT and GPE to be retrieved with a single ioctl.
526                  * So we try to read the GPT and the GPE using that old style
527                  * ioctl.
528                  */
529                 dk_ioc.dki_lba = 1;
530                 dk_ioc.dki_length = label_len;
531                 rval = check_label(fd, &dk_ioc);
```

```
532         }

534         if (rval < 0) {
535                 free(efi);
536                 return (rval);
537         }

539         /* LINTED -- always longlong aligned */
540         efi_parts = (efi_gpe_t *)(((char *)efi) + disk_info.dki_lbsize);

542         /*
543          * Assemble this into a "dk_gpt" struct for easier
544          * digestibility by applications.
545          */
546         vtoc->efi_version = LE_32(efi->efi_gpt_Revision);
547         vtoc->efi_nparts = LE_32(efi->efi_gpt_NumberOfPartitionEntries);
548         vtoc->efi_part_size = LE_32(efi->efi_gpt_SizeOfPartitionEntry);
549         vtoc->efi_lbasize = disk_info.dki_lbsize;
550         vtoc->efi_last_lba = disk_info.dki_capacity - 1;
551         vtoc->efi_first_u_lba = LE_64(efi->efi_gpt_FirstUsableLBA);
552         vtoc->efi_last_u_lba = LE_64(efi->efi_gpt_LastUsableLBA);
553         vtoc->efi_altern_lba = LE_64(efi->efi_gpt_AlternateLBA);
554         UUID_LE_CONVERT(vtoc->efi_disk_uguid, efi->efi_gpt_DiskGUID);

556         /*
557          * If the array the user passed in is too small, set the length
558          * to what it needs to be and return
559          */
560         if (user_length < vtoc->efi_nparts) {
561                 return (VT_EINVAL);
562         }

564         for (i = 0; i < vtoc->efi_nparts; i++) {

566                 UUID_LE_CONVERT(vtoc->efi_parts[i].p_guid,
567                     efi_parts[i].efi_gpe_PartitionTypeGUID);

569                 for (j = 0;
570                     j < sizeof (conversion_array)
571                     / sizeof (struct uuid_to_ptag); j++) {

573                         if (bcmp(&vtoc->efi_parts[i].p_guid,
574                             &conversion_array[j].uuid,
575                             sizeof (struct uuid)) == 0) {
576                                 vtoc->efi_parts[i].p_tag =
577                                     conversion_array[j].p_tag;
578                                 break;
579                         }
580                 }
581                 if (vtoc->efi_parts[i].p_tag == V_UNASSIGNED)
582                         continue;
583                 vtoc->efi_parts[i].p_flag =
584                     LE_16(efi_parts[i].efi_gpe_Attributes.PartitionAttrs);
585                 vtoc->efi_parts[i].p_start =
586                     LE_64(efi_parts[i].efi_gpe_StartingLBA);
587                 vtoc->efi_parts[i].p_size =
588                     LE_64(efi_parts[i].efi_gpe_EndingLBA) -
589                     vtoc->efi_parts[i].p_start + 1;
590                 for (j = 0; j < EFI_PART_NAME_LEN; j++) {
591                         vtoc->efi_parts[i].p_name[j] =
592                             (uchar_t)LE_16(
593                             efi_parts[i].efi_gpe_PartitionName[j]);
594                 }

596                 UUID_LE_CONVERT(vtoc->efi_parts[i].p_uguid,
597                     efi_parts[i].efi_gpe_UniquePartitionGUID);
```



```
598         }
599         free(efi);

601         return (dki_info.dki_partition);
602 }
_____unchanged_portion_omitted_


974 /*
975  * write EFI label and backup label
976  */
977 int
978 efi_write(int fd, struct dk_gpt *vtoc)
979 {
980         dk_efi_t                dk_ioc;
981         efi_gpt_t               *efi;
982         efi_gpe_t               *efi_parts;
983         int                     i, j;
984         struct dk_cinfo         dki_info;
985         int                     nblocks;
986         diskaddr_t              lba_backup_gpt_hdr;

988         if (ioctl(fd, DKIOCINFO, (caddr_t)&dki_info) == -1) {
989                 if (efi_debug)
990                         (void) fprintf(stderr, "DKIOCINFO errno 0x%x\n", errno);
991                 switch (errno) {
992                 case EIO:
993                         return (VT_EIO);
994                 case EINVAL:
995                         return (VT_EINVAL);
996                 default:
997                         return (VT_ERROR);
998                 }
999         }

1001        if (check_input(vtoc))
1002                return (VT_EINVAL);

1004        dk_ioc.dki_lba = 1;
1005        if (NBLOCKS(vtoc->efi_nparts, vtoc->efi_lbasize) < 34) {
1006                dk_ioc.dki_length = EFI_MIN_ARRAY_SIZE + vtoc->efi_lbasize;
1007        } else {
1008                dk_ioc.dki_length = NBLOCKS(vtoc->efi_nparts,
1009                    vtoc->efi_lbasize) *
1010                    vtoc->efi_lbasize;
1011        }

1013        /*
1014         * the number of blocks occupied by GUID partition entry array
1015         */
1016        nblocks = dk_ioc.dki_length / vtoc->efi_lbasize - 1;

1018        /*
1019         * Backup GPT header is located on the block after GUID
1020         * partition entry array. Here, we calculate the address
1021         * for backup GPT header.
1022         */
1023        lba_backup_gpt_hdr = vtoc->efi_last_u_lba + 1 + nblocks;
1024        if ((dk_ioc.dki_data = calloc(1, dk_ioc.dki_length)) == NULL)
1023        if ((dk_ioc.dki_data = calloc(dk_ioc.dki_length, 1)) == NULL)
1025                return (VT_ERROR);

1027        efi = dk_ioc.dki_data;

1029        /* stuff user's input into EFI struct */
1030        efi->efi_gpt_Signature = LE_64(EFI_SIGNATURE);
```

```
1031         efi->efi_gpt_Revision = LE_32(vtoc->efi_version); /* 0x02000100 */
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);

1043         /* LINTED -- always longlong aligned */
1044         efi_parts = (efi_gpe_t *)((char *)dk_ioc.dki_data + vtoc->efi_lbasize);

1046         for (i = 0; i < vtoc->efi_nparts; i++) {
1047                 for (j = 0;
1048                     j < sizeof (conversion_array) /
1049                     sizeof (struct uuid_to_ptag); j++) {

1051                         if (vtoc->efi_parts[i].p_tag ==
1052                             conversion_array[j].p_tag) {
1053                                 UUID_LE_CONVERT(
1054                                     efi_parts[i].efi_gpe_PartitionTypeGUID,
1055                                     conversion_array[j].uuid);
1056                                 break;
1057                         }
1058                 }

1060                 if (j == sizeof (conversion_array) /
1061                     sizeof (struct uuid_to_ptag)) {
1062                         /*
1063                          * If we didn't have a matching uuid match, bail here.
1064                          * Don't write a label with unknown uuid.
1065                          */
1066                         if (efi_debug) {
1067                                 (void) fprintf(stderr,
1068                                     "Unknown uuid for p_tag %d\n",
1069                                     vtoc->efi_parts[i].p_tag);
1070                         }
1071                         return (VT_EINVAL);
1072                 }

1074                 efi_parts[i].efi_gpe_StartingLBA =
1075                     LE_64(vtoc->efi_parts[i].p_start);
1076                 efi_parts[i].efi_gpe_EndingLBA =
1077                     LE_64(vtoc->efi_parts[i].p_start +
1078                     vtoc->efi_parts[i].p_size - 1);
1079                 efi_parts[i].efi_gpe_Attributes.PartitionAttrs =
1080                     LE_16(vtoc->efi_parts[i].p_flag);
1081                 for (j = 0; j < EFI_PART_NAME_LEN; j++) {
1082                         efi_parts[i].efi_gpe_PartitionName[j] =
1083                             LE_16((ushort_t)vtoc->efi_parts[i].p_name[j]);
1084                 }
1085                 if ((vtoc->efi_parts[i].p_tag != V_UNASSIGNED) &&
1086                     uuid_is_null((uchar_t *)&vtoc->efi_parts[i].p_uguid)) {
1087                         (void) uuid_generate((uchar_t *)
1088                             &vtoc->efi_parts[i].p_uguid);
1089                 }
1090                 bcopy(&vtoc->efi_parts[i].p_uguid,
1091                     &efi_parts[i].efi_gpe_UniquePartitionGUID,
1092                     sizeof (uuid_t));
1093         }
1094         efi->efi_gpt_PartitionEntryArrayCRC32 =
1095             LE_32(efi_crc32((unsigned char *)efi_parts,
1096             vtoc->efi_nparts * (int)sizeof (struct efi_gpe)));
```



```
1097         efi->efi_gpt_HeaderCRC32 =
1098             LE_32(efi_crc32((unsigned char *)efi, sizeof (struct efi_gpt)));

1100         if (efi_ioctl(fd, DKIOCSETEFI, &dk_ioc) == -1) {
1101                 free(dk_ioc.dki_data);
1102                 switch (errno) {
1103                 case EIO:
1104                         return (VT_EIO);
1105                 case EINVAL:
1106                         return (VT_EINVAL);
1107                 default:
1108                         return (VT_ERROR);
1109                 }
1110         }

1112         /* write backup partition array */
1113         dk_ioc.dki_lba = vtoc->efi_last_u_lba + 1;
1114         dk_ioc.dki_length -= vtoc->efi_lbasize;
1115         /* LINTED */
1116         dk_ioc.dki_data = (efi_gpt_t *)((char *)dk_ioc.dki_data +
1117             vtoc->efi_lbasize);

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

1148         if (efi_ioctl(fd, DKIOCSETEFI, &dk_ioc) == -1) {
1149                 if (efi_debug) {
1150                         (void) fprintf(stderr,
1151                             "write of backup header to block %llu failed, "
1152                             "errno %d\n",
1153                             lba_backup_gpt_hdr,
1154                             errno);
1155                 }
1156         }
1157         /* write the PMBR */
1158         (void) write_pmbr(fd, vtoc);
1159         free(dk_ioc.dki_data);
1160         return (0);
1161 }
```

_____***unchanged_portion_omitted_***

****
5497 Thu Jan 17 14:25:04 2019
new/usr/src/lib/libilb/common/ilb_comm.c
10100 Illumos is confused about calloc() arguments
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright (c) 2018, Joyent, Inc.
 */

#include <stdlib.h>
#include <strings.h>
#include <unistd.h>
#include <stddef.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <thread.h>
#include <synch.h>
#include <libilb_impl.h>
#include <libilb.h>

/* Assertion: the calling thread has a hold on the handle */
static void
i_ilb_socket_set_err(ilb_handle_t h, ilb_status_t err)
{
        ilb_handle_impl_t       *hi = (ilb_handle_impl_t *)h;

        if (h == ILB_INVALID_HANDLE)
                return;
        hi->h_valid = B_FALSE;
        hi->h_error = err;
}

ilb_status_t
ilb_open(ilb_handle_t *hp)
{
        ilb_handle_impl_t       *hi = NULL;
        int                     s = -1;
        struct sockaddr_un sa = {AF_UNIX, SOCKET_PATH};
        ilb_status_t            rc = ILB_STATUS_OK;
        int                     sobufsz;
```



```
        if (hp == NULL)
                return (ILB_STATUS_EINVAL);

        hi = calloc(1, sizeof (*hi));
        hi = calloc(sizeof (*hi), 1);
        if (hi == NULL)
                return (ILB_STATUS_ENOMEM);

        if (cond_init(&hi->h_cv, USYNC_THREAD, NULL) != 0) {
                rc = ILB_STATUS_INTERNAL;
                goto out;
        }

        if (mutex_init(&hi->h_lock, USYNC_THREAD | LOCK_ERRORCHECK, NULL)
            != 0) {
                rc = ILB_STATUS_INTERNAL;
                goto out;
        }

        hi->h_busy = B_FALSE;

        if ((s = socket(PF_UNIX, SOCK_SEQPACKET, 0)) == -1 ||
            connect(s, (struct sockaddr *)&sa, sizeof (sa.sun_path))
            == -1) {
                rc = ILB_STATUS_SOCKET;
                goto out;
        }

        /* The socket buffer must be at least the max size of a message */
        sobufsz = ILBD_MSG_SIZE;
        if (setsockopt(s, SOL_SOCKET, SO_SNDBUF, &sobufsz,
            sizeof (sobufsz)) != 0) {
                rc = ILB_STATUS_SOCKET;
                (void) close(s);
                goto out;
        }
        if (setsockopt(s, SOL_SOCKET, SO_RCVBUF, &sobufsz,
            sizeof (sobufsz)) != 0) {
                rc = ILB_STATUS_SOCKET;
                (void) close(s);
                goto out;
        }

        hi->h_socket = s;
        hi->h_valid = B_TRUE;

out:
        if (rc != ILB_STATUS_OK && s != -1)
                (void) close(s);

        if (rc == ILB_STATUS_OK) {
                *hp = (ilb_handle_t)hi;
        } else {
                free(hi);
                *hp = ILB_INVALID_HANDLE;
        }
        return (rc);
}
```

_____***unchanged_portion_omitted_***

****
3887 Thu Jan 17 14:25:04 2019
new/usr/src/lib/libipmi/common/libipmi.c
10100 Illumos is confused about calloc() arguments
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright (c) 2018, Joyent, Inc.
 */

#include <libipmi.h>
#include <string.h>

#include "ipmi_impl.h"

ipmi_handle_t *
ipmi_open(int *errp, char **msg, uint_t xport_type, nvlist_t *params)
{
        ipmi_handle_t *ihp;
        static char errmsg[48];

        if (msg)
                *msg = NULL;

        if ((ihp = calloc(1, sizeof (ipmi_handle_t))) == NULL) {
        if ((ihp = calloc(sizeof (ipmi_handle_t), 1)) == NULL) {
                *errp = EIPMI_NOMEM;
                if (msg)
                        *msg = "memory allocation failure";
                return (NULL);
        }

        switch (xport_type) {
        case IPMI_TRANSPORT_BMC:
                ihp->ih_transport = &ipmi_transport_bmc;
                break;
        case IPMI_TRANSPORT_LAN:
                ihp->ih_transport = &ipmi_transport_lan;
                break;
        default:
                *msg = "invalid transport type specified";
                return (NULL);
        }
```



```
        ihp->ih_retries = 3;

        if ((ihp->ih_tdata = ihp->ih_transport->it_open(ihp, params)) == NULL ||
            ipmi_sdr_init(ihp) != 0 || ipmi_entity_init(ihp) != 0) {
                *errp = ihp->ih_errno;
                if (msg) {
                        (void) strncpy(errmsg, ipmi_errmsg(ihp), 47);
                        errmsg[47] = '\0';
                        *msg = errmsg;
                }
                ipmi_close(ihp);
                return (NULL);
        }

        return (ihp);
}
```

_____***unchanged_portion_omitted_***