```
**********************************************************
   34175 Wed Jan 21 10:03:51 2015
new/usr/src/lib/sun_sas/common/devtree_device_disco.c
5552 libsun_sas leaks devids
Reviewed by: Alek Pinchuk <alek.pinchuk@nexenta.com>
Reviewed by: Jean McCormack <jean.mccormack@nexenta.com>
Reviewed by: Marcel Telka <marcel.telka@nexenta.com>
**********************************************************
```

```
/*
 * 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 2015 Nexenta Systems, Inc.  All rights reserved.
#endif /* ! codereview */
 */

#include        <sun_sas.h>
#include        <sys/types.h>
#include        <netinet/in.h>
#include        <inttypes.h>
#include        <ctype.h>
#include        <sys/scsi/scsi_address.h>
#include        <libdevid.h>

/*
 * Get the preferred minor node for the given path.
 * ":n" for tapes, ":c,raw" for disks,
 * and ":0" for enclosures.
 */
static void
get_minor(char *devpath, char *minor)
{
        const char      ROUTINE[] = "get_minor";
        char    fullpath[MAXPATHLEN];
        int     fd;

        if ((strstr(devpath, "/st@")) || (strstr(devpath, "/tape@"))) {
                (void) strcpy(minor, ":n");
        } else if (strstr(devpath, "/smp@")) {
                (void) strcpy(minor, ":smp");
        } else if ((strstr(devpath, "/ssd@")) || (strstr(devpath, "/sd@")) ||
            (strstr(devpath, "/disk@"))) {
                (void) strcpy(minor, ":c,raw");
        } else if ((strstr(devpath, "/ses@")) || (strstr(devpath,
            "/enclosure@"))) {
                (void) snprintf(fullpath, MAXPATHLEN, "%s%s%s", DEVICES_DIR,
```



```
                    devpath, ":0");
                /* reset errno to 0 */
                errno = 0;
                if ((fd = open(fullpath, O_RDONLY)) == -1) {
                        /*
                         * :0 minor doesn't exist. assume bound to sgen driver
                         * and :ses minor exist.
                         */
                        if (errno == ENOENT) {
                                (void) strcpy(minor, ":ses");
                        }
                } else {
                        (void) strcpy(minor, ":0");
                        (void) close(fd);
                }
        } else {
                log(LOG_DEBUG, ROUTINE, "Unrecognized target (%s)",
                    devpath);
                minor[0] = '\0';
        }

}

/*
 * Free the attached port allocation.
 */
static void
free_attached_port(struct sun_sas_port *port_ptr)
{
        struct sun_sas_port     *tgt_port, *last_tgt_port;
        struct ScsiEntryList    *scsi_info = NULL, *last_scsi_info = NULL;

        tgt_port = port_ptr->first_attached_port;
        while (tgt_port != NULL) {
                /* Free target mapping data list first. */
                scsi_info = tgt_port->scsiInfo;
                while (scsi_info != NULL) {
                        last_scsi_info = scsi_info;
                        scsi_info = scsi_info->next;
                        free(last_scsi_info);
                }
                last_tgt_port = tgt_port;
                tgt_port = tgt_port->next;
                free(last_tgt_port->port_attributes.\
                    PortSpecificAttribute.SASPort);
                free(last_tgt_port);
        }

        port_ptr->first_attached_port = NULL;
        port_ptr->port_attributes.PortSpecificAttribute.\
            SASPort->NumberofDiscoveredPorts = 0;
}

/*
 * Fill domainPortWWN.
 * should be called after completing discovered port discovery.
 */
void
fillDomainPortWWN(struct sun_sas_port *port_ptr)
{
        const char    ROUTINE[] = "fillDomainPortWWN";
        struct sun_sas_port *disco_port_ptr;
        struct phy_info *phy_ptr;
        uint64_t    domainPort = 0;
        struct ScsiEntryList        *mapping_ptr;
```

```
125         for (disco_port_ptr = port_ptr->first_attached_port;
126             disco_port_ptr != NULL; disco_port_ptr = disco_port_ptr->next) {
127                 if (disco_port_ptr->port_attributes.PortType ==
128                     HBA_PORTTYPE_SASEXPANDER &&
129                     wwnConversion(disco_port_ptr->port_attributes.
130                     PortSpecificAttribute.SASPort->
131                     AttachedSASAddress.wwn) ==
132                     wwnConversion(port_ptr->port_attributes.
133                     PortSpecificAttribute.SASPort->
134                     LocalSASAddress.wwn)) {
135                         (void) memcpy(&domainPort,
136                             disco_port_ptr->port_attributes.
137                             PortSpecificAttribute.
138                             SASPort->LocalSASAddress.wwn, 8);
139                         break;
140                 }
141         }

143         if (domainPort == 0) {
144                 if (port_ptr->first_attached_port) {
145                         /*
146                          * there is no expander device attached on an HBA port
147                          * domainPortWWN should not stay to 0 since multiple
148                          * hba ports can have the same LocalSASAddres within
149                          * the same HBA.
150                          * Set the SAS address of direct attached target.
151                          */
152                         if (wwnConversion(port_ptr->port_attributes.
153                             PortSpecificAttribute.SASPort->
154                             LocalSASAddress.wwn) ==
155                             wwnConversion(port_ptr->first_attached_port->
156                             port_attributes.PortSpecificAttribute.
157                             SASPort->AttachedSASAddress.wwn)) {
158                                 (void) memcpy(&domainPort,
159                                     port_ptr->first_attached_port->
160                                     port_attributes.PortSpecificAttribute.
161                                     SASPort->LocalSASAddress.wwn, 8);
162                         } else {
163                                 /*
164                                  * SAS address is not upstream connected.
165                                  * domainPortWWN stays as 0.
166                                  */
167                                 log(LOG_DEBUG, ROUTINE,
168                                     "DomainPortWWN is not set. "
169                                     "Device(s) are visible on the HBA port "
170                                     "but there is no expander or directly "
171                                     "attached port with matching upsteam "
172                                     "attached SAS address for "
173                                     "HBA port (Local SAS Address: %016llx).",
174                                     wwnConversion(port_ptr->port_attributes.
175                                     PortSpecificAttribute.
176                                     SASPort->LocalSASAddress.wwn));
177                                 return;
178                         }
179                 } else {
180                         /*
181                          * There existss an iport without properly configured
182                          * child smp ndoes or  child node or pathinfo.
183                          * domainPortWWN stays as 0.
184                          */
185                         log(LOG_DEBUG, ROUTINE,
186                             "DomainPortWWN is not set.  No properly "
187                             "configured smp or directly attached port "
188                             "found on HBA port(Local SAS Address: %016llx).",
189                             wwnConversion(port_ptr->port_attributes.
190                             PortSpecificAttribute.
```



```
191                             SASPort->LocalSASAddress.wwn));
192                         return;
193                 }
194         }

196         /* fill up phy info */
197         for (phy_ptr = port_ptr->first_phy; phy_ptr != NULL;
198             phy_ptr = phy_ptr->next) {
199                 (void) memcpy(phy_ptr->phy.domainPortWWN.wwn, &domainPort, 8);
200         }

202         /* fill up target mapping */
203         for (disco_port_ptr = port_ptr->first_attached_port;
204             disco_port_ptr != NULL; disco_port_ptr = disco_port_ptr->next) {
205                 for (mapping_ptr = disco_port_ptr->scsiInfo;
206                     mapping_ptr != NULL;
207                     mapping_ptr = mapping_ptr->next) {
208                         (void) memcpy(mapping_ptr->entry.PortLun.
209                             domainPortWWN.wwn, &domainPort, 8);
210                 }
211         }
212 }

214 /*
215  * Finds attached device(target) from devinfo node.
216  */
217 static HBA_STATUS
218 get_attached_devices_info(di_node_t node, struct sun_sas_port *port_ptr)
219 {
220         const char                  ROUTINE[] = "get_attached_devices_info";
221         char                        *propStringData = NULL;
222         int                         *propIntData = NULL;
223         int64_t                     *propInt64Data = NULL;
224         scsi_lun_t                  samLun;
225         ddi_devid_t                 devid;
226         char                        *guidStr;
227         char                        *unit_address;
228         char                        *charptr;
229         char                        *devpath, link[MAXNAMELEN];
230         char                        fullpath[MAXPATHLEN+1];
231         char                        minorname[MAXNAMELEN+1];
232         struct ScsiEntryList        *mapping_ptr;
233         HBA_WWN                     SASAddress, AttachedSASAddress;
234         struct sun_sas_port         *disco_port_ptr;
235         uint_t                      state = 0;
236         int                         portfound, rval, size;
237         int                         port_state = HBA_PORTSTATE_ONLINE;
238         uint64_t                    tmpAddr;

240         if (port_ptr == NULL) {
241                 log(LOG_DEBUG, ROUTINE, "NULL port_ptr argument");
242                 return (HBA_STATUS_ERROR);
243         }

245         if ((devpath = di_devfs_path(node)) == NULL) {
246                 log(LOG_DEBUG, ROUTINE,
247                     "Device in device tree has no path. Skipping.");
248                 return (HBA_STATUS_ERROR);
249         }

251         if ((di_instance(node) == -1) || di_retired(node)) {
252                 log(LOG_DEBUG, ROUTINE,
253                     "dev node (%s) returned instance of -1 or is retired. "
254                     " Skipping.", devpath);
255                 di_devfs_path_free(devpath);
256                 return (HBA_STATUS_OK);
```

```
257         }
258         state = di_state(node);
259         /* when node is not attached and online, set the state to offline. */
260         if (((state & DI_DRIVER_DETACHED) == DI_DRIVER_DETACHED) ||
261             ((state & DI_DEVICE_OFFLINE) == DI_DEVICE_OFFLINE)) {
262                 log(LOG_DEBUG, ROUTINE,
263                     "dev node (%s) is either OFFLINE or DETACHED",
264                     devpath);
265                 port_state = HBA_PORTSTATE_OFFLINE;
266         }
267
268         /* add the "/devices" in the begining at the end */
269         (void) snprintf(fullpath, sizeof (fullpath), "%s%s",
270             DEVICES_DIR, devpath);
271
272         (void) memset(&SASAddress, 0, sizeof (SASAddress));
273         if ((unit_address = di_bus_addr(node)) != NULL) {
274                 if ((charptr = strchr(unit_address, ',')) != NULL) {
275                         *charptr = '\0';
276                 }
277                 for (charptr = unit_address; *charptr != '\0'; charptr++) {
278                         if (isxdigit(*charptr)) {
279                                 break;
280                         }
281                 }
282                 if (*charptr != '\0') {
283                         tmpAddr = htonll(strtoll(charptr, NULL, 16));
284                         (void) memcpy(&SASAddress.wwn[0], &tmpAddr, 8);
285                 } else {
286                         log(LOG_DEBUG, ROUTINE,
287                             "No proper target port info on unit address of %s",
288                             fullpath);
289                         di_devfs_path_free(devpath);
290                         return (HBA_STATUS_ERROR);
291                 }
292         } else {
293                 log(LOG_DEBUG, ROUTINE,
294                     "Fail to get unit address of %s.",
295                     fullpath);
296                 di_devfs_path_free(devpath);
297                 return (HBA_STATUS_ERROR);
298         }
299
300         (void) memset(&AttachedSASAddress, 0, sizeof (AttachedSASAddress));
301         if (di_prop_lookup_strings(DDI_DEV_T_ANY, node, "attached-port",
302             &propStringData) != -1) {
303                 for (charptr = propStringData; *charptr != '\0'; charptr++) {
304                         if (isxdigit(*charptr)) {
305                                 break;
306                         }
307                 }
308                 if (*charptr != '\0') {
309                         tmpAddr = htonll(strtoll(charptr, NULL, 16));
310                         (void) memcpy(AttachedSASAddress.wwn, &tmpAddr, 8);
311                         /* check the attached address of hba port. */
312                         if (memcmp(port_ptr->port_attributes.
313                             PortSpecificAttribute.SASPort->LocalSASAddress.wwn,
314                             &tmpAddr, 8) == 0) {
315                                 /*
316                                  * When attached-port is set from iport
317                                  * attached-port prop, we do the cross check
318                                  * with device's own SAS address.
319                                  *
320                                  * If not set, we store device's own SAS
321                                  * address to iport attached SAS address.
322                                  */
```



```
323                                 if (wwnConversion(port_ptr->port_attributes.
324                                     PortSpecificAttribute.SASPort->
325                                     AttachedSASAddress.wwn)) {
326                                         /* verify the Attaached SAS Addr. */
327                                         if (memcmp(port_ptr->port_attributes.
328                                             PortSpecificAttribute.SASPort->
329                                             AttachedSASAddress.wwn,
330                                             SASAddress.wwn, 8) != 0) {
331                                 /* indentation move begin. */
332                                 log(LOG_DEBUG, ROUTINE,
333                                     "iport attached-port(%016llx) do not"
334                                     " match with level 1 Local"
335                                     " SAS address(%016llx).",
336                                     wwnConversion(port_ptr->port_attributes.
337                                     PortSpecificAttribute.
338                                     SASPort->AttachedSASAddress.wwn),
339                                     wwnConversion(SASAddress.wwn));
340                                 di_devfs_path_free(devpath);
341                                 free_attached_port(port_ptr);
342                                 return (HBA_STATUS_ERROR);
343                                 /* indentation move ends. */
344                                         }
345                                 } else {
346                                         (void) memcpy(port_ptr->port_attributes.
347                                             PortSpecificAttribute.
348                                             SASPort->AttachedSASAddress.wwn,
349                                             &SASAddress.wwn[0], 8);
350                                 }
351                         }
352                 } else {
353                         log(LOG_DEBUG, ROUTINE,
354                             "No proper attached SAS address value on device %s",
355                             fullpath);
356                         di_devfs_path_free(devpath);
357                         free_attached_port(port_ptr);
358                         return (HBA_STATUS_ERROR);
359                 }
360         } else {
361                 log(LOG_DEBUG, ROUTINE,
362                     "Property AttachedSASAddress not found for device \"%s\"",
363                     fullpath);
364                 di_devfs_path_free(devpath);
365                 free_attached_port(port_ptr);
366                 return (HBA_STATUS_ERROR);
367         }
368
369         /*
370          * walk the disco list to make sure that there isn't a matching
371          * port and node wwn or a matching device path
372          */
373         portfound = 0;
374         for (disco_port_ptr = port_ptr->first_attached_port;
375             disco_port_ptr != NULL;
376             disco_port_ptr = disco_port_ptr->next) {
377                 if ((disco_port_ptr->port_attributes.PortState !=
378                     HBA_PORTSTATE_ERROR) && (memcmp(disco_port_ptr->
379                     port_attributes.PortSpecificAttribute.
380                     SASPort->LocalSASAddress.wwn, SASAddress.wwn, 8) == 0)) {
381                         /*
382                          * found matching disco_port
383                          * look for matching device path
384                          */
385                         portfound = 1;
386                         for (mapping_ptr = disco_port_ptr->scsiInfo;
387                             mapping_ptr != NULL;
388                             mapping_ptr = mapping_ptr->next) {
```

```
389                                 if (strstr(mapping_ptr-> entry.ScsiId.
390                                     OSDeviceName, devpath) != 0) {
391                                         log(LOG_DEBUG, ROUTINE,
392                                             "Found an already discovered "
393                                             "device %s.", fullpath);
394                                         di_devfs_path_free(devpath);
395                                         return (HBA_STATUS_OK);
396                                 }
397                         }
398                         if (portfound == 1) {
399                                 break;
400                         }
401                 }
402         }

404         if (portfound == 0) {
405                 /*
406                  * there are no matching SAS address.
407                  * this must be a new device
408                  */
409                 if ((disco_port_ptr = (struct sun_sas_port *)calloc(1,
410                     sizeof (struct sun_sas_port))) == NULL)  {
411                         OUT_OF_MEMORY(ROUTINE);
412                         di_devfs_path_free(devpath);
413                         free_attached_port(port_ptr);
414                         return (HBA_STATUS_ERROR);
415                 }

417                 if ((disco_port_ptr->port_attributes.PortSpecificAttribute.\
418                     SASPort = (struct SMHBA_SAS_Port *)calloc(1,
419                     sizeof (struct SMHBA_SAS_Port))) == NULL) {
420                         OUT_OF_MEMORY("add_hba_port_info");
421                         di_devfs_path_free(devpath);
422                         free_attached_port(port_ptr);
423                         return (HBA_STATUS_ERROR);
424                 }

426                 (void) memcpy(disco_port_ptr->port_attributes.
427                     PortSpecificAttribute.SASPort->LocalSASAddress.wwn,
428                     SASAddress.wwn, 8);
429                 (void) memcpy(disco_port_ptr->port_attributes.
430                     PortSpecificAttribute.SASPort->AttachedSASAddress.wwn,
431                     AttachedSASAddress.wwn, 8);

433                 /* Default to unknown until we figure out otherwise */
434                 rval = di_prop_lookup_strings(DDI_DEV_T_ANY, node,
435                     "variant", &propStringData);
436                 if (rval < 0) {
437                         /* check if it is SMP target */
438                         charptr = di_driver_name(node);
439                         if (charptr != NULL && (strncmp(charptr, "smp",
440                             strlen(charptr)) == 0)) {
441                                 disco_port_ptr->port_attributes.PortType =
442                                     HBA_PORTTYPE_SASEXPANDER;
443                                 disco_port_ptr->port_attributes.
444                                     PortSpecificAttribute.
445                                     SASPort->PortProtocol =
446                                     HBA_SASPORTPROTOCOL_SMP;
447                                 if (lookupSMPLink(devpath, (char *)link) ==
448                                     HBA_STATUS_OK) {
449                 /* indentation changed here. */
450                 (void) strlcpy(disco_port_ptr->port_attributes.
451                     OSDeviceName, link,
452                     sizeof (disco_port_ptr->port_attributes.OSDeviceName));
453                 /* indentation change ends here. */
454                                 } else {
```



```
455                 /* indentation changed here. */
456                 get_minor(devpath, minorname);
457                 (void) snprintf(fullpath, sizeof (fullpath), "%s%s%s",
458                     DEVICES_DIR, devpath, minorname);
459                 (void) strlcpy(disco_port_ptr->port_attributes.
460                     OSDeviceName, fullpath,
461                     sizeof (disco_port_ptr->port_attributes.OSDeviceName));
462                 /* indentation change ends here. */
463                                 }
464                         } else {
465                                 disco_port_ptr->port_attributes.PortType =
466                                     HBA_PORTTYPE_SASDEVICE;
467                                 disco_port_ptr->port_attributes.\
468                                     PortSpecificAttribute.\
469                                     SASPort->PortProtocol =
470                                     HBA_SASPORTPROTOCOL_SSP;
471                         }
472                 } else {
473                         if ((strcmp(propStringData, "sata") == 0) ||
474                             (strcmp(propStringData, "atapi") == 0)) {
475                                 disco_port_ptr->port_attributes.PortType =
476                                     HBA_PORTTYPE_SATADEVICE;
477                                 disco_port_ptr->port_attributes.\
478                                     PortSpecificAttribute.SASPort->PortProtocol
479                                     = HBA_SASPORTPROTOCOL_SATA;
480                         } else {
481                                 log(LOG_DEBUG, ROUTINE,
482                                     "Unexpected variant prop value %s found on",
483                                     " device %s", propStringData, fullpath);
484                                 /*
485                                  * Port type will be 0
486                                  * which is not valid type.
487                                  */
488                         }
489                 }

491                 /* SMP device was handled already */
492                 if (disco_port_ptr->port_attributes.OSDeviceName[0] == '\0') {
493                 /* indentation change due to ctysle check on sizeof. */
494                 size = sizeof (disco_port_ptr->port_attributes.OSDeviceName);
495                         (void) strlcpy(disco_port_ptr->port_attributes.
496                             OSDeviceName, fullpath, size);
497                 }

499                 /* add new discovered port into the list */

501                 if (port_ptr->first_attached_port == NULL) {
502                         port_ptr->first_attached_port = disco_port_ptr;
503                         disco_port_ptr->index = 0;
504                         port_ptr->port_attributes.PortSpecificAttribute.\
505                             SASPort->NumberofDiscoveredPorts = 1;
506                 } else {
507                         disco_port_ptr->next = port_ptr->first_attached_port;
508                         port_ptr->first_attached_port = disco_port_ptr;
509                         disco_port_ptr->index = port_ptr->port_attributes.\
510                             PortSpecificAttribute.\
511                             SASPort->NumberofDiscoveredPorts;
512                         port_ptr->port_attributes.PortSpecificAttribute.\
513                             SASPort->NumberofDiscoveredPorts++;
514                 }
515                 disco_port_ptr->port_attributes.PortState = port_state;
516         }

518         if (disco_port_ptr->port_attributes.PortType ==
519             HBA_PORTTYPE_SASEXPANDER) {
520             /* No mapping data for expander device.  return ok here. */
```

```
521                 di_devfs_path_free(devpath);
522                 return (HBA_STATUS_OK);
523         }

525         if ((mapping_ptr = (struct ScsiEntryList *)calloc
526                 (1, sizeof (struct ScsiEntryList))) == NULL) {
527                 OUT_OF_MEMORY(ROUTINE);
528                 di_devfs_path_free(devpath);
529                 free_attached_port(port_ptr);
530                 return (HBA_STATUS_ERROR);
531         }

533         if (di_prop_lookup_ints(DDI_DEV_T_ANY, node, "lun",
534             &propIntData) != -1) {
535                 mapping_ptr->entry.ScsiId.ScsiOSLun = *propIntData;
536         } else {
537                 if ((charptr = strchr(unit_address, ',')) != NULL) {
538                         charptr++;
539                         mapping_ptr->entry.ScsiId.ScsiOSLun =
540                             strtoull(charptr, NULL, 10);
541                 } else {
542                         log(LOG_DEBUG, ROUTINE,
543                             "Failed to get LUN from the unit address of device "
544                             " %s.", fullpath);
545                         di_devfs_path_free(devpath);
546                         free_attached_port(port_ptr);
547                         return (HBA_STATUS_ERROR);
548                 }
549         }

551         /* get TargetLun(SAM-LUN). */
552         if (di_prop_lookup_int64(DDI_DEV_T_ANY, node, "lun64",
553             &propInt64Data) != -1) {
554                 samLun = scsi_lun64_to_lun(*propInt64Data);
555                 (void) memcpy(&mapping_ptr->entry.PortLun.TargetLun,
556                     &samLun, 8);
557         } else {
558                 log(LOG_DEBUG, "get_attached_devices_info",
559                     "No lun64 prop found on device %s.", fullpath);
560                 di_devfs_path_free(devpath);
561                 free_attached_port(port_ptr);
562                 return (HBA_STATUS_ERROR);
563         }

565         if (di_prop_lookup_ints(DDI_DEV_T_ANY, node,
566             "target", &propIntData) != -1) {
567                 mapping_ptr->entry.ScsiId.ScsiTargetNumber = *propIntData;
568         } else {
569                 mapping_ptr->entry.ScsiId.ScsiTargetNumber = di_instance(node);
570         }

572         /* get ScsiBusNumber */
573         mapping_ptr->entry.ScsiId.ScsiBusNumber = port_ptr->cntlNumber;

575         (void) memcpy(mapping_ptr->entry.PortLun.PortWWN.wwn,
576             SASAddress.wwn, 8);

578         /* Store the devices path for now.  We'll convert to /dev later */
579         get_minor(devpath, minorname);
580         (void) snprintf(mapping_ptr->entry.ScsiId.OSDeviceName,
581             sizeof (mapping_ptr->entry.ScsiId.OSDeviceName),
582             "%s%s%s", DEVICES_DIR, devpath, minorname);

584         /* reset errno to 0 */
585         errno = 0;
586         if (di_prop_lookup_strings(DDI_DEV_T_ANY, node, "devid",
```



```
587             &propStringData) != -1) {
588                 if (devid_str_decode(propStringData, &devid, NULL) != -1) {
589                         guidStr = devid_to_guid(devid);
590                         if (guidStr != NULL) {
591                                 (void) strlcpy(mapping_ptr->entry.LUID.buffer,
592                                     guidStr,
593                                     sizeof (mapping_ptr->entry.LUID.buffer));
 25                                  guidStr, 256);
594                                 devid_free_guid(guidStr);
595                         } else {
596                                 /*
597                                  * Note:
598                                  * if logical unit associated page 83 id
599                                  * descriptor is not avaialble for the device
600                                  * devid_to_guid returns NULL with errno 0.
 32                                 * devid_to_guid returns NULl with errno 0.
601                                  */
602                                 log(LOG_DEBUG, ROUTINE,
603                                     "failed to get devid guid on (%s) : %s",
604                                     devpath, strerror(errno));
605                         }

607                         devid_free(devid);
608 #endif /* ! codereview */
609                 } else {
610                         /*
611                          * device may not support proper page 83 id descriptor.
612                          * leave LUID attribute to NULL and continue.
613                          */
614                         log(LOG_DEBUG, ROUTINE,
615                             "failed to decode devid prop on (%s) : %s",
616                             devpath, strerror(errno));
617                 }
618         } else {
619                 /* leave LUID attribute to NULL and continue. */
620                 log(LOG_DEBUG, ROUTINE,
621                     "failed to get devid prop on (%s) : %s",
622                     devpath, strerror(errno));
623         }

625         if (disco_port_ptr->scsiInfo == NULL) {
626                 disco_port_ptr->scsiInfo = mapping_ptr;
627         } else {
628                 mapping_ptr->next = disco_port_ptr->scsiInfo;
629                 disco_port_ptr->scsiInfo = mapping_ptr;
630         }

632         di_devfs_path_free(devpath);

634         return (HBA_STATUS_OK);
635 }

637 /*
638  * Finds attached device(target) from pathinfo node.
639  */
640 static HBA_STATUS
641 get_attached_paths_info(di_path_t path, struct sun_sas_port *port_ptr)
642 {
643         char                        ROUTINE[] = "get_attached_paths_info";
644         char                        *propStringData = NULL;
645         int                         *propIntData = NULL;
646         int64_t                     *propInt64Data = NULL;
647         scsi_lun_t                  samLun;
648         ddi_devid_t                 devid;
649         char                        *guidStr;
650         char                        *unit_address;
```

```
651         char                        *charptr;
652         char                        *clientdevpath = NULL;
653         char                        *pathdevpath = NULL;
654         char                        fullpath[MAXPATHLEN+1];
655         char                        minorname[MAXNAMELEN+1];
656         struct ScsiEntryList        *mapping_ptr;
657         HBA_WWN                     SASAddress, AttachedSASAddress;
658         struct sun_sas_port         *disco_port_ptr;
659         di_path_state_t             state = 0;
660         di_node_t                   clientnode;
661         int                         portfound, size;
662         int                         port_state = HBA_PORTSTATE_ONLINE;
663         uint64_t                    tmpAddr;

665         if (port_ptr == NULL) {
666                 log(LOG_DEBUG, ROUTINE, "NULL port_ptr argument");
667                 return (HBA_STATUS_ERROR);
668         }

670         /* if not null, free before return. */
671         pathdevpath = di_path_devfs_path(path);

673         state = di_path_state(path);
674         /* when node is not attached and online, set the state to offline. */
675         if ((state == DI_PATH_STATE_OFFLINE) ||
676             (state == DI_PATH_STATE_FAULT)) {
677                 log(LOG_DEBUG, ROUTINE,
678                     "path node (%s) is either OFFLINE or FAULT state",
679                     pathdevpath ?  pathdevpath : "(missing device path)");
680                 port_state = HBA_PORTSTATE_OFFLINE;
681         }

683         if (clientnode = di_path_client_node(path)) {
684                 if (di_retired(clientnode)) {
685                         log(LOG_DEBUG, ROUTINE,
686                             "client node of path (%s) is retired. Skipping.",
687                             pathdevpath ?  pathdevpath :
688                             "(missing device path)");
689                         if (pathdevpath) di_devfs_path_free(pathdevpath);
690                         return (HBA_STATUS_OK);
691                 }
692                 if ((clientdevpath = di_devfs_path(clientnode)) == NULL) {
693                         log(LOG_DEBUG, ROUTINE,
694                             "Client device of path (%s) has no path. Skipping.",
695                             pathdevpath ?  pathdevpath :
696                             "(missing device path)");
697                         if (pathdevpath) di_devfs_path_free(pathdevpath);
698                         return (HBA_STATUS_ERROR);
699                 }
700         } else {
701                 log(LOG_DEBUG, ROUTINE,
702                     "Failed to get client device from a path (%s).",
703                     pathdevpath ?  pathdevpath :
704                     "(missing device path)");
705                 if (pathdevpath) di_devfs_path_free(pathdevpath);
706                 return (HBA_STATUS_ERROR);
707         }

709         /* add the "/devices" in the begining and the :devctl at the end */
710         (void) snprintf(fullpath, sizeof (fullpath), "%s%s", DEVICES_DIR,
711             clientdevpath);

713         (void) memset(&SASAddress, 0, sizeof (SASAddress));
714         if ((unit_address = di_path_bus_addr(path)) != NULL) {
715                 if ((charptr = strchr(unit_address, ',')) != NULL) {
716                         *charptr = '\0';
```



```
717                 }
718                 for (charptr = unit_address; *charptr != '\0'; charptr++) {
719                         if (isxdigit(*charptr)) {
720                                 break;
721                         }
722                 }
723                 if (charptr != '\0') {
724                         tmpAddr = htonll(strtoll(charptr, NULL, 16));
725                         (void) memcpy(&SASAddress.wwn[0], &tmpAddr, 8);
726                 } else {
727                         log(LOG_DEBUG, ROUTINE,
728                             "No proper target port info on unit address of "
729                             "path (%s).", pathdevpath ?  pathdevpath :
730                             "(missing device path)");
731                         if (pathdevpath) di_devfs_path_free(pathdevpath);
732                         di_devfs_path_free(clientdevpath);
733                         return (HBA_STATUS_ERROR);
734                 }
735         } else {
736                 log(LOG_DEBUG, ROUTINE, "Fail to get unit address of path(%s).",
737                     "path (%s).", pathdevpath ?  pathdevpath :
738                     "(missing device path)");
739                 if (pathdevpath) di_devfs_path_free(pathdevpath);
740                 di_devfs_path_free(clientdevpath);
741                 return (HBA_STATUS_ERROR);
742         }

744         (void) memset(&AttachedSASAddress, 0, sizeof (AttachedSASAddress));
745         if (di_path_prop_lookup_strings(path, "attached-port",
746             &propStringData) != -1) {
747                 for (charptr = propStringData; *charptr != '\0'; charptr++) {
748                         if (isxdigit(*charptr)) {
749                                 break;
750                         }
751                 }
752                 if (*charptr != '\0') {
753                         tmpAddr = htonll(strtoll(charptr, NULL, 16));
754                         (void) memcpy(AttachedSASAddress.wwn, &tmpAddr, 8);
755                         /*  check the attached address of hba port. */
756                         if (memcmp(port_ptr->port_attributes.
757                             PortSpecificAttribute.SASPort->
758                             LocalSASAddress.wwn, &tmpAddr, 8) == 0) {
759                                 if (wwnConversion(port_ptr->port_attributes.
760                                     PortSpecificAttribute.SASPort->
761                                     AttachedSASAddress.wwn)) {
762                                         /* verify the attaached SAS Addr. */
763                                         if (memcmp(port_ptr->port_attributes.
764                                             PortSpecificAttribute.SASPort->
765                                             AttachedSASAddress.wwn,
766                                             SASAddress.wwn, 8) != 0) {
767                                 /* indentation move begin. */
768                                 log(LOG_DEBUG, ROUTINE,
769                                     "iport attached-port(%016llx) do not"
770                                     " match with level 1 Local"
771                                     " SAS address(%016llx).",
772                                     wwnConversion(port_ptr->port_attributes.
773                                     PortSpecificAttribute.
774                                     SASPort->AttachedSASAddress.wwn),
775                                     wwnConversion(SASAddress.wwn));
776                                 if (pathdevpath)
777                                         di_devfs_path_free(pathdevpath);
778                                 di_devfs_path_free(clientdevpath);
779                                 free_attached_port(port_ptr);
780                                 return (HBA_STATUS_ERROR);
781                                 /* indentation move ends. */
782                                         }
```

```
783                                 } else {
784                                         /* store the Attaached SAS Addr. */
785                                         (void) memcpy(port_ptr->port_attributes.
786                                             PortSpecificAttribute.
787                                             SASPort->AttachedSASAddress.wwn,
788                                             &SASAddress.wwn[0], 8);
789                                 }
790                         }
791                 } else {
792                         log(LOG_DEBUG, ROUTINE,
793                             "No proper attached SAS address value of path (%s)",
794                             pathdevpath ?  pathdevpath :
795                             "(missing device path)");
796                         if (pathdevpath) di_devfs_path_free(pathdevpath);
797                         di_devfs_path_free(clientdevpath);
798                         free_attached_port(port_ptr);
799                         return (HBA_STATUS_ERROR);
800                 }
801         } else {
802                 log(LOG_DEBUG, ROUTINE,
803                     "Property attached-port not found for path (%s)",
804                     pathdevpath ?  pathdevpath :
805                     "(missing device path)");
806                 if (pathdevpath) di_devfs_path_free(pathdevpath);
807                 di_devfs_path_free(clientdevpath);
808                 free_attached_port(port_ptr);
809                 return (HBA_STATUS_ERROR);
810         }

812         /*
813          * walk the disco list to make sure that there isn't a matching
814          * port and node wwn or a matching device path
815          */
816         portfound = 0;
817         for (disco_port_ptr = port_ptr->first_attached_port;
818             disco_port_ptr != NULL;
819             disco_port_ptr = disco_port_ptr->next) {
820                 if ((disco_port_ptr->port_attributes.PortState !=
821                     HBA_PORTSTATE_ERROR) &&
822                     (memcmp(disco_port_ptr->port_attributes.
823                     PortSpecificAttribute.SASPort->LocalSASAddress.wwn,
824                     SASAddress.wwn, 8) == 0)) {
825                         /*
826                          * found matching disco_port
827                          * look for matching device path
828                          */
829                         portfound = 1;
830                         for (mapping_ptr = disco_port_ptr->scsiInfo;
831                             mapping_ptr != NULL;
832                             mapping_ptr = mapping_ptr->next) {
833                                 if (strstr(mapping_ptr-> entry.ScsiId.
834                                     OSDeviceName, clientdevpath) != 0) {
835                                         log(LOG_DEBUG, ROUTINE,
836                                             "Found an already discovered "
837                                             "device %s.", clientdevpath);
838                                         if (pathdevpath)
839                                                 di_devfs_path_free(pathdevpath);
840                                         di_devfs_path_free(clientdevpath);
841                                         return (HBA_STATUS_OK);
842                                 }
843                         }
844                         if (portfound == 1) {
845                                 break;
846                         }
847                 }
848         }
```



```
850         if (portfound == 0) {
851                 /*
852                  * there are no matching SAS address.
853                  * this must be a new device
854                  */
855                 if ((disco_port_ptr = (struct sun_sas_port *)calloc(1,
856                                     sizeof (struct sun_sas_port))) == NULL)  {
857                         OUT_OF_MEMORY(ROUTINE);
858                         if (pathdevpath) di_devfs_path_free(pathdevpath);
859                         di_devfs_path_free(clientdevpath);
860                         free_attached_port(port_ptr);
861                         return (HBA_STATUS_ERROR);
862                 }

864                 if ((disco_port_ptr->port_attributes.PortSpecificAttribute.\
865                     SASPort = (struct SMHBA_SAS_Port *)calloc(1,
866                     sizeof (struct SMHBA_SAS_Port))) == NULL) {
867                         OUT_OF_MEMORY("add_hba_port_info");
868                         if (pathdevpath) di_devfs_path_free(pathdevpath);
869                         di_devfs_path_free(clientdevpath);
870                         free_attached_port(port_ptr);
871                         return (HBA_STATUS_ERROR);
872                 }

874                 (void) memcpy(disco_port_ptr->port_attributes.
875                     PortSpecificAttribute.
876                     SASPort->LocalSASAddress.wwn, SASAddress.wwn, 8);
877                 (void) memcpy(disco_port_ptr->port_attributes.
878                     PortSpecificAttribute.
879                     SASPort->AttachedSASAddress.wwn, AttachedSASAddress.wwn, 8);

881                 /* Default to unknown until we figure out otherwise */
882                 if (di_path_prop_lookup_strings(path, "variant",
883                     &propStringData) != -1) {
884                         if ((strcmp(propStringData, "sata") == 0) ||
885                             (strcmp(propStringData, "atapi") == 0)) {
886                                 disco_port_ptr->port_attributes.PortType =
887                                     HBA_PORTTYPE_SATADEVICE;
888                                 disco_port_ptr->port_attributes.\
889                                     PortSpecificAttribute.SASPort->PortProtocol
890                                     = HBA_SASPORTPROTOCOL_SATA;
891                         } else {
892                                 log(LOG_DEBUG, ROUTINE,
893                                     "Unexpected variant prop value %s found on",
894                                     " path (%s)", propStringData,
895                                     pathdevpath ?  pathdevpath :
896                                     "(missing device path)");
897                                 /*
898                                  * Port type will be 0
899                                  * which is not valid type.
900                                  */
901                         }
902                 } else {
903                         disco_port_ptr->port_attributes.PortType =
904                             HBA_PORTTYPE_SASDEVICE;
905                         disco_port_ptr->port_attributes.PortSpecificAttribute.\
906                             SASPort->PortProtocol = HBA_SASPORTPROTOCOL_SSP;
907                 }

909                 if (disco_port_ptr->port_attributes.OSDeviceName[0] == '\0') {
910                 /* indentation change due to ctysle check on sizeof. */
911                 size = sizeof (disco_port_ptr->port_attributes.OSDeviceName);
912                         if (pathdevpath != NULL) {
913                                 (void) strlcpy(disco_port_ptr->port_attributes.
914                                     OSDeviceName, pathdevpath, size);
```

```
915                         }
916                 }

918                 /* add new discovered port into the list */
919                 if (port_ptr->first_attached_port == NULL) {
920                         port_ptr->first_attached_port = disco_port_ptr;
921                         disco_port_ptr->index = 0;
922                         port_ptr->port_attributes.PortSpecificAttribute.\
923                             SASPort->NumberofDiscoveredPorts = 1;
924                 } else {
925                         disco_port_ptr->next = port_ptr->first_attached_port;
926                         port_ptr->first_attached_port = disco_port_ptr;
927                         disco_port_ptr->index = port_ptr->port_attributes.\
928                             PortSpecificAttribute.\
929                             SASPort->NumberofDiscoveredPorts;
930                         port_ptr->port_attributes.PortSpecificAttribute.\
931                             SASPort->NumberofDiscoveredPorts++;
932                 }
933                 disco_port_ptr->port_attributes.PortState = port_state;
934         }

936         if ((mapping_ptr = (struct ScsiEntryList *)calloc
937                 (1, sizeof (struct ScsiEntryList))) == NULL) {
938                 OUT_OF_MEMORY(ROUTINE);
939                 if (pathdevpath) di_devfs_path_free(pathdevpath);
940                 di_devfs_path_free(clientdevpath);
941                 free_attached_port(port_ptr);
942                 return (HBA_STATUS_ERROR);
943         }

945         if (di_path_prop_lookup_ints(path, "lun", &propIntData) != -1) {
946                 mapping_ptr->entry.ScsiId.ScsiOSLun = *propIntData;
947         } else {
948                 if ((charptr = strchr(unit_address, ',')) != NULL) {
949                         charptr++;
950                         mapping_ptr->entry.ScsiId.ScsiOSLun =
951                             strtoull(charptr, NULL, 10);
952                 } else {
953                         log(LOG_DEBUG, ROUTINE,
954                             "Failed to get LUN from unit address of path(%s).",
955                             pathdevpath ?  pathdevpath :
956                             "(missing device path)");
957                         if (pathdevpath) di_devfs_path_free(pathdevpath);
958                         di_devfs_path_free(clientdevpath);
959                         free_attached_port(port_ptr);
960                         return (HBA_STATUS_ERROR);
961                 }
962         }

964         /* Get TargetLun(SAM LUN). */
965         if (di_path_prop_lookup_int64s(path, "lun64", &propInt64Data) != -1) {
966                 samLun = scsi_lun64_to_lun(*propInt64Data);
967                 (void) memcpy(&mapping_ptr->entry.PortLun.TargetLun,
968                     &samLun, 8);
969         } else {
970                 log(LOG_DEBUG, ROUTINE, "No lun64 prop found on path (%s)",
971                     pathdevpath ?  pathdevpath :
972                     "(missing device path)");
973                 if (pathdevpath) di_devfs_path_free(pathdevpath);
974                 di_devfs_path_free(clientdevpath);
975                 free_attached_port(port_ptr);
976                 return (HBA_STATUS_ERROR);
977         }

979         if (di_path_prop_lookup_ints(path, "target", &propIntData) != -1) {
980                 mapping_ptr->entry.ScsiId.ScsiTargetNumber = *propIntData;
```



```
981         } else {
982                 mapping_ptr->entry.ScsiId.ScsiTargetNumber =
983                     di_path_instance(path);
984         }

986         /* get ScsiBusNumber */
987         mapping_ptr->entry.ScsiId.ScsiBusNumber = port_ptr->cntlNumber;

989         (void) memcpy(mapping_ptr->entry.PortLun.PortWWN.wwn,
990             SASAddress.wwn, 8);

992         /* Store the devices path for now.  We'll convert to /dev later */
993         get_minor(clientdevpath, minorname);
994         (void) snprintf(mapping_ptr->entry.ScsiId.OSDeviceName,
995             sizeof (mapping_ptr->entry.ScsiId.OSDeviceName),
996             "%s%s%s", DEVICES_DIR, clientdevpath, minorname);

998         /* get luid. */
999         errno = 0; /* reset errno to 0 */
1000        if (di_prop_lookup_strings(DDI_DEV_T_ANY, clientnode, "devid",
1001            &propStringData) != -1) {
1002                if (devid_str_decode(propStringData, &devid, NULL) != -1) {
1003                        guidStr = devid_to_guid(devid);
1004                        if (guidStr != NULL) {
1005                                (void) strlcpy(mapping_ptr->entry.LUID.buffer,
1006                                    guidStr,
1007                                    sizeof (mapping_ptr->entry.LUID.buffer));
1008                                devid_free_guid(guidStr);
1009                        } else {
1010                                /*
1011                                 * Note:
1012                                 * if logical unit associated page 83 id
1013                                 * descriptor is not avaialble for the device
1014                                 * devid_to_guid returns NULL with errno 0.
  38                             * devid_to_guid returns NULl with errno 0.
1015                                 */
1016                                log(LOG_DEBUG, ROUTINE,
1017                                    "failed to get devid guid on (%s)",
1018                                    " associated with path(%s) : %s",
1019                                    clientdevpath,
1020                                    pathdevpath ?  pathdevpath :
1021                                    "(missing device path)",
1022                                    strerror(errno));
1023                        }

1025                        devid_free(devid);
1026 #endif /* ! codereview */
1027                } else {
1028                        /*
1029                         * device may not support proper page 83 id descriptor.
1030                         * leave LUID attribute to NULL and continue.
1031                         */
1032                        log(LOG_DEBUG, ROUTINE,
1033                            "failed to decode devid prop on (%s)",
1034                            " associated with path(%s) : %s",
1035                            clientdevpath,
1036                            pathdevpath ?  pathdevpath :
1037                            "(missing device path)",
1038                            strerror(errno));
1039                }
1040        } else {
1041                /* leave LUID attribute to NULL and continue. */
1042                log(LOG_DEBUG, ROUTINE, "Failed to get devid on %s"
1043                    " associated with path(%s) : %s", clientdevpath,
1044                    pathdevpath ?  pathdevpath : "(missing device path)",
1045                    strerror(errno));
```

```
1046         }

1048         if (disco_port_ptr->scsiInfo == NULL) {
1049                 disco_port_ptr->scsiInfo = mapping_ptr;
1050         } else {
1051                 mapping_ptr->next = disco_port_ptr->scsiInfo;
1052                 disco_port_ptr->scsiInfo = mapping_ptr;
1053         }

1055         if (pathdevpath) di_devfs_path_free(pathdevpath);
1056         di_devfs_path_free(clientdevpath);

1058         return (HBA_STATUS_OK);
1059 }

1061 /*
1062  * walks the devinfo tree retrieving all hba information
1063  */
1064 extern HBA_STATUS
1065 devtree_attached_devices(di_node_t node, struct sun_sas_port *port_ptr)
1066 {
1067         const char              ROUTINE[] = "devtree_attached_devices";
1068         di_node_t               nodechild = DI_NODE_NIL;
1069         di_path_t               path = DI_PATH_NIL;

1071         /* child should be device */
1072         if ((nodechild = di_child_node(node)) == DI_NODE_NIL) {
1073                 log(LOG_DEBUG, ROUTINE,
1074                     "No devinfo child on the HBA port node.");
1075         }

1077         if ((path = di_path_phci_next_path(node, path)) ==
1078             DI_PATH_NIL) {
1079                 log(LOG_DEBUG, ROUTINE,
1080                     "No pathinfo node on the HBA port node.");
1081         }

1083         if ((nodechild == DI_NODE_NIL) && (path == DI_PATH_NIL)) {
1084                 return (HBA_STATUS_OK);
1085         }

1087         while (nodechild != DI_NODE_NIL) {
1088                 if (get_attached_devices_info(nodechild, port_ptr)
1089                     != HBA_STATUS_OK) {
1090                         break;
1091                 }
1092                 nodechild = di_sibling_node(nodechild);
1093         }


1096         while (path != DI_PATH_NIL) {
1097                 if (get_attached_paths_info(path, port_ptr)
1098                     != HBA_STATUS_OK) {
1099                         break;
1100                 }
1101                 path = di_path_phci_next_path(node, path);
1102         }

1104         return (HBA_STATUS_OK);
1105 }
```