```
**********************************************************
   14768 Tue Aug 18 18:50:58 2015
new/usr/src/cmd/modload/plcysubr.c
6139 help gcc figure out variable initialization
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License, Version 1.0 only
   6  * (the "License").  You may not use this file except in compliance
   7  * with the License.
   8  *
   9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10  * or http://www.opensolaris.org/os/licensing.
  11  * See the License for the specific language governing permissions
  12  * and limitations under the License.
  13  *
  14  * When distributing Covered Code, include this CDDL HEADER in each
  15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16  * If applicable, add the following below this CDDL HEADER, with the
  17  * fields enclosed by brackets "[]" replaced with your own identifying
  18  * information: Portions Copyright [yyyy] [name of copyright owner]
  19  *
  20  * CDDL HEADER END
  21  */
  22 /*
  23  * Copyright 2003 Sun Microsystems, Inc.  All rights reserved.
  24  * Use is subject to license terms.
  25  *
  26  * Device policy specific subroutines.  We cannot merge them with
  27  * drvsubr.c because of static linking requirements.
  28  */

  30 #pragma ident   "%Z%%M% %I%     %E% SMI"

  30 #include <stdio.h>
  31 #include <stdlib.h>
  32 #include <unistd.h>
  33 #include <string.h>
  34 #include <ctype.h>
  35 #include <priv.h>
  36 #include <string.h>
  37 #include <libgen.h>
  38 #include <libintl.h>
  39 #include <errno.h>
  40 #include <alloca.h>
  41 #include <sys/modctl.h>
  42 #include <sys/devpolicy.h>
  43 #include <sys/stat.h>
  44 #include <sys/sysmacros.h>

  46 #include "addrem.h"
  47 #include "errmsg.h"
  48 #include "plcysubr.h"

  50 size_t devplcysys_sz;
  51 const priv_impl_info_t *privimplinfo;

  53 /*
  54  * New token types should be parsed in parse_plcy_entry.
  55  */
  56 #define PSET    0

  58 typedef struct token {
  59         const char      *token;
```



```
  60         int             type;
  61         ptrdiff_t       off;
  62 } token_t;
______unchanged_portion_omitted_

 247 static int
 248 delete_one_entry(const char *filename, const char *entry)
 249 {
 250         char tfile[MAXPATHLEN];
 251         char ofile[MAXPATHLEN];
 252         char *nfile;
 253         FILE *old, *new;
 254         fileentry_t *fep;
 255         struct stat buf;
 256         int newfd;
 257         char *mpart;
 258         boolean_t delall;
 259         boolean_t delrange;
 260         minor_t rlo, rhi;
 261         char rtype;

 263         mpart = strchr(entry, ':');
 264         if (mpart == NULL) {
 265                 delall = B_TRUE;
 266                 delrange = B_FALSE;
 267         } else {
 268                 delall = B_FALSE;
 269                 mpart++;
 270                 if (*mpart == '(') {
 271                         if (parse_minor_range(mpart, &rlo, &rhi, &rtype) != 0)
 272                                 return (-1);
 273                         delrange = B_TRUE;
 274                 } else {
 275                         delrange = B_FALSE;
 276                 }
 277         }

 279         if (strlen(filename) + sizeof (XEND)  > sizeof (tfile))
 280                 return (-1);

 282         old = fopen(filename, "r");

 284         if (old == NULL)
 285                 return (-1);

 287         (void) snprintf(tfile, sizeof (tfile), "%s%s", filename, XEND);
 288         (void) snprintf(ofile, sizeof (ofile), "%s%s", filename, ".old");

 290         nfile = mktemp(tfile);

 292         new = fopen(nfile, "w");
 293         if (new == NULL) {
 294                 (void) fclose(old);
 295                 return (ERROR);
 296         }

 298         newfd = fileno(new);

 300         /* Copy permissions, ownership */
 301         if (fstat(fileno(old), &buf) == 0) {
 302                 (void) fchown(newfd, buf.st_uid, buf.st_gid);
 303                 (void) fchmod(newfd, buf.st_mode);
 304         } else {
 305                 (void) fchown(newfd, 0, 3);     /* root:sys */
 306                 (void) fchmod(newfd, 0644);
 307         }
```

```
309         while ((fep = fgetline(old))) {
310                 char *tok;
311                 char *min;
312                 char *tail;
313                 char tc;
314                 int len;
315
316                 /* Trailing comments */
317                 if (fep->entry == NULL) {
318                         (void) fputs(fep->rawbuf, new);
319                         break;
320                 }
321
322                 tok = fep->entry;
323                 while (*tok && isspace(*tok))
324                         tok++;
325
326                 if (*tok == '\0') {
327                         (void) fputs(fep->rawbuf, new);
328                         break;
329                 }
330
331                 /* Make sure we can recover the remainder incl. whitespace */
332                 tail = strpbrk(tok, "\t\n ");
333                 if (tail == NULL)
334                         tail = tok + strlen(tok);
335                 tc = *tail;
336                 *tail = '\0';
337
340                 if (delall || delrange) {
338                 min = strchr(tok, ':');
339                 if (min && (delall || delrange))
342                         if (min)
340                         *min++ = '\0';
344                 }
341
342                 len = strlen(tok);
343                 if (delrange) {
344                         minor_t lo, hi;
345                         char type;
346
347                         /*
348                          * Delete or shrink overlapping ranges.
349                          */
350                         if (strncmp(entry, tok, len) == 0 &&
351                             entry[len] == ':' &&
352                             min != NULL &&
353                             parse_minor_range(min, &lo, &hi, &type) == 0 &&
354                             (type == rtype || rtype == '\0') &&
355                             lo <= rhi && hi >= rlo) {
356                                 minor_t newlo, newhi;
357
358                                 /* Complete overlap, then drop it. */
359                                 if (lo >= rlo && hi <= rhi)
360                                         continue;
361
362                                 /* Partial overlap, shrink range */
363                                 if (lo < rlo)
364                                         newhi = rlo - 1;
365                                 else
366                                         newhi = hi;
367                                 if (hi > rhi)
368                                         newlo = rhi + 1;
369                                 else
370                                         newlo = lo;
```



```
372                                 /* restore NULed character */
373                                 *tail = tc;
374
375                                 /* Split range? */
376                                 if (newlo > newhi) {
377                                         /*
378                                          * We have two ranges:
379                                          * lo ... newhi (== rlo - 1)
380                                          * newlo (== rhi + 1) .. hi
381                                          */
382                                         put_minor_range(new, fep, tok, tail,
383                                             lo, newhi, type);
384                                         put_minor_range(new, NULL, tok, tail,
385                                             newlo, hi, type);
386                                 } else {
387                                         put_minor_range(new, fep, tok, tail,
388                                             newlo, newhi, type);
389                                 }
390                                 continue;
391                         }
392                 } else if (strcmp(entry, tok) == 0 ||
393                     (strncmp(entry, tok, len) == 0 &&
394                     entry[len] == ':' &&
395                     entry[len+1] == '*' &&
396                     entry[len+2] == '\0')) {
397                         /*
398                          * Delete exact match.
399                          */
400                         continue;
401                 }
402
403                 /* Copy unaffected entry. */
404                 (void) fputs(fep->rawbuf, new);
405         }
406         (void) fclose(old);
407         (void) fflush(new);
408         (void) fsync(newfd);
409         if (ferror(new) == 0 && fclose(new) == 0 && fep != NULL) {
410                 if (rename(filename, ofile) != 0) {
411                         perror(NULL);
412                         (void) fprintf(stderr, gettext(ERR_UPDATE), ofile);
413                         (void) unlink(ofile);
414                         (void) unlink(nfile);
415                         return (ERROR);
416                 } else if (rename(nfile, filename) != 0) {
417                         perror(NULL);
418                         (void) fprintf(stderr, gettext(ERR_UPDATE), ofile);
419                         (void) rename(ofile, filename);
420                         (void) unlink(nfile);
421                         return (ERROR);
422                 }
423                 (void) unlink(ofile);
424         } else
425                 (void) unlink(nfile);
426         return (0);
427 }
```

_____***unchanged_portion_omitted_***

****
27771 Tue Aug 18 18:50:58 2015
new/usr/src/common/iscsit/iscsit_common.c
6139 help gcc figure out variable initialization
****
_____***unchanged_portion_omitted_***

```
781 #ifndef _KERNEL
782 int
783 it_tpg_to_nv(it_tpg_t *tpg, nvlist_t **nvl)
784 {
785         int             ret;
786         char            **portalArray = NULL;
787         int             i;
788         it_portal_t     *ptr;

790         if (!nvl) {
791                 return (EINVAL);
792         }

794         if (!tpg) {
795                 /* nothing to do */
796                 return (0);
797         }

799         ret = nvlist_alloc(nvl, NV_UNIQUE_NAME, 0);
800         if (ret != 0) {
801                 return (ret);
802         }

804         ret = nvlist_add_uint64(*nvl, "generation", tpg->tpg_generation);

806         if ((ret == 0) && tpg->tpg_portal_list) {
807                 /* add the portals */
808                 portalArray = iscsit_zalloc(tpg->tpg_portal_count *
809                     sizeof (it_portal_t));
810                 if (portalArray == NULL) {
811                         nvlist_free(*nvl);
812                         *nvl = NULL;
813                         return (ENOMEM);
814                 }

816                 i = 0;
817                 ptr = tpg->tpg_portal_list;

819                 while (ptr && (i < tpg->tpg_portal_count)) {
820                         ret = sockaddr_to_str(&(ptr->portal_addr),
821                             &(portalArray[i]));
822                         if (ret != 0) {
823                                 break;
824                         }
825                         ptr = ptr->portal_next;
826                         i++;
827                 }
828         }

829                 if (ret == 0) {
830         if ((ret == 0) && portalArray) {
830                         ret = nvlist_add_string_array(*nvl, "portalList",
831                             portalArray, i);
832                 }


836         if (portalArray) {
835                 while (--i >= 0) {
836                         if (portalArray[i]) {
```



```
837                                 iscsit_free(portalArray[i],
838                                     strlen(portalArray[i] + 1));
839                         }
840                 }
841                 iscsit_free(portalArray,
842                     tpg->tpg_portal_count * sizeof (it_portal_t));
843         }

845         if (ret != 0) {
846                 nvlist_free(*nvl);
847                 *nvl = NULL;
848         }

850         return (ret);
851 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   28076 Tue Aug 18 18:50:58 2015
new/usr/src/lib/auditd_plugins/binfile/binfile.c
6139 help gcc figure out variable initialization
**********************************************************
______unchanged_portion_omitted_

 248 /*
 249  * create a linked list of directories available for writing
 250  *
 251  * if a list already exists, the two are compared and the new one is
 252  * used only if it is different than the old.
 253  *
 254  * returns -2 for new or changed list, 0 for unchanged list and -1 for
 255  * error.  (Positive returns are for AUDITD_<error code> values)
 256  *
 257  */
 258 static int
 259 loadauditlist(char *dirstr, char *minfreestr)
 260 {
 261         dirlist_t       *n1, *n2;
 262         dirlist_t       *listhead = NULL;
 263         dirlist_t       *thisdir;
 264         int             node_count = 0;
 265         int             rc;
 266         int             temp_minfree;

 268         static dirlist_t        *activeList = NULL;     /* directory list */

 270         DPRINT((dbfp, "binfile: Loading audit list from audit service "
 271             "(audit_binfile)\n"));

 273         if (dirstr == NULL || minfreestr == NULL) {
 274                 DPRINT((dbfp, "binfile: internal error"));
 275                 return (-1);
 276         }
 277         if ((rc = growauditlist(&listhead, dirstr, NULL, &node_count)) != 0) {
 278                 return (rc);
 279         }
 280         if (node_count == 0) {
 281                 /*
 282                  * there was a problem getting the directory
 283                  * list or remote host info from the audit_binfile
 284                  * configuration even though auditd thought there was
 285                  * at least 1 good entry
 286                  */
 287                 DPRINT((dbfp, "binfile: "
 288                     "problem getting directory / libpath list "
 289                     "from audit_binfile configuration.\n"));
 290                 return (-1);
 291         }

 293 #if DEBUG
 294         /* print out directory list */
 295         if (listhead != NULL) {
 296                 (void) fprintf(dbfp, "Directory list:\n\t%s\n",
 297                     listhead->dl_dirname);
 298                 thisdir = listhead->dl_next;

 300                 while (thisdir != listhead) {
 301                         (void) fprintf(dbfp, "\t%s\n", thisdir->dl_dirname);
 302                         thisdir = thisdir->dl_next;
 303                 }
 304         }
 305 #endif  /* DEBUG */
```



```
 307         thisdir = listhead;

 309         /* See if the list has changed (rc = 0 if no change, else 1) */
 310         rc = 0;
 311         if (node_count == activeCount) {
 312                 n1 = listhead;
 313                 n2 = activeList;
 314                 do {
 315                         if (strcmp(n1->dl_dirname, n2->dl_dirname) != 0) {
 316                                 DPRINT((dbfp,
 317                                     "binfile: new dirname = %s\n"
 318                                     "binfile: old dirname = %s\n",
 319                                     n1->dl_dirname,
 320                                     n2->dl_dirname));
 321                                 rc = -2;
 322                                 break;
 323                         }
 324                         n1 = n1->dl_next;
 325                         n2 = n2->dl_next;
 326                 } while ((n1 != listhead) && (n2 != activeList));
 327         } else {
 328                 DPRINT((dbfp, "binfile: dir counts differs\n"
 329                     "binfile:  old dir count = %d\n"
 330                     "binfile:  new dir count = %d\n",
 331                     activeCount, node_count));
 332                 rc = -2;
 333         }
 334         if (rc == -2) {
 335                 (void) pthread_mutex_lock(&log_mutex);
 336                 DPRINT((dbfp, "loadauditlist:  close / open audit.log(4)\n"));
 337                 if (open_log(listhead) == 0) {
 338                         openNewFile = 1;        /* try again later */
 339                 } else {
 340                         openNewFile = 0;
 341                 }
 342                 freedirlist(activeList);        /* old list */
 343                 activeList = listhead;          /* new list */
 344                 activeDir = startdir = thisdir;
 345                 activeCount = node_count;
 346                 (void) pthread_mutex_unlock(&log_mutex);
 347         } else {
 348                 freedirlist(listhead);
 349         }

 351         /* Get the minfree value. */
 352         if (minfreestr != NULL)
 352         temp_minfree = atoi(minfreestr);

 354         if ((temp_minfree < 0) || (temp_minfree > 100))
 355                 temp_minfree = 0;

 357         if (minfree != temp_minfree) {
 358                 DPRINT((dbfp, "minfree:  old = %d, new = %d\n",
 359                     minfree, temp_minfree));
 360                 rc = -2;                /* data change */
 361                 minfree = temp_minfree;
 362         }

 364         return (rc);
 365 }
______unchanged_portion_omitted_
```

****
30478 Tue Aug 18 18:50:58 2015
new/usr/src/lib/libhotplug/common/libhotplug.c
6139 help gcc figure out variable initialization
****
_____***unchanged_portion_omitted_***

```
433 /*
434  * hp_path()
435  *
436  *      Return the path (and maybe connection name) of a node.
437  *      The caller must supply two buffers, each MAXPATHLEN size.
438  */
439 int
440 hp_path(hp_node_t node, char *path, char *connection)
441 {
442         hp_node_t       root;
443         hp_node_t       parent;
444         int             i;
445         char            *s;
446         char            components[MAXPATHLEN];

448         i_hp_dprintf("hp_path: node=%p, path=%p, connection=%p\n", (void *)node,
449             (void *)path, (void *)connection);

451         if ((node == NULL) || (path == NULL) || (connection == NULL)) {
452                 i_hp_dprintf("hp_path: invalid arguments.\n");
453                 return (EINVAL);
454         }

456         (void) memset(path, 0, MAXPATHLEN);
457         (void) memset(connection, 0, MAXPATHLEN);
458         (void) memset(components, 0, MAXPATHLEN);

460         /*  Set 'connection' only for connectors and ports */
461         if ((node->hp_type == HP_NODE_CONNECTOR) ||
462             (node->hp_type == HP_NODE_PORT))
463                 (void) strlcpy(connection, node->hp_name, MAXPATHLEN);

465         /* Trace back to the root node, accumulating components */
466         for (parent = node, root = parent; parent != NULL;
467             root = parent, parent = parent->hp_parent) {
466         for (parent = node; parent != NULL; parent = parent->hp_parent) {
468                 if (parent->hp_type == HP_NODE_DEVICE) {
469                         (void) strlcat(components, "/", MAXPATHLEN);
470                         (void) strlcat(components, parent->hp_name, MAXPATHLEN);
471                 }
471                 if (parent->hp_parent == NULL)
472                         root = parent;
472         }

474         /* Ensure the snapshot actually contains a base path */
475         if (root->hp_basepath == NULL) {
476                 i_hp_dprintf("hp_path: missing base pathname.\n");
477                 return (EFAULT);
478         }

480         /*
481          * Construct the path.  Start with the base path from the root
482          * node, then append the accumulated components in reverse order.
483          */
484         if (strcmp(root->hp_basepath, "/") != 0) {
485                 (void) strlcat(path, root->hp_basepath, MAXPATHLEN);
486                 if ((root->hp_type == HP_NODE_DEVICE) &&
487                     ((s = strrchr(path, '/')) != NULL))
488                         *s = '\0';
```



```
489         }
490         for (i = strlen(components) - 1; i >= 0; i--) {
491                 if (components[i] == '/') {
492                         (void) strlcat(path, &components[i], MAXPATHLEN);
493                         components[i] = '\0';
494                 }
495         }

497         return (0);
498 }
```

_____***unchanged_portion_omitted_***