```
**********************************************************
   72161 Wed Oct 17 22:00:49 2012
new/usr/src/lib/libsldap/common/ns_connect.c
3285 memory leaks in libsldap
**********************************************************
```

```
/*
 * 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) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
 * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
 */

#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <synch.h>
#include <time.h>
#include <libintl.h>
#include <thread.h>
#include <syslog.h>
#include <sys/mman.h>
#include <nsswitch.h>
#include <nss_dbdefs.h>
#include "solaris-priv.h"
#include "solaris-int.h"
#include "ns_sldap.h"
#include "ns_internal.h"
#include "ns_cache_door.h"
#include "ns_connmgmt.h"
#include "ldappr.h"
#include <sys/stat.h>
#include <fcntl.h>
#include <procfs.h>
#include <unistd.h>

#define USE_DEFAULT_PORT 0

static ns_ldap_return_code performBind(const ns_cred_t *,
                                       LDAP *,
                                       int,
                                       ns_ldap_error_t **,
                                       int,
                                       int);
static ns_ldap_return_code createSession(const ns_cred_t *,
                                       const char *,
                                       uint16_t,
```



```
                                       int,
                                       LDAP **,
                                       ns_ldap_error_t **);

extern int ldap_sasl_cram_md5_bind_s(LDAP *, char *, struct berval *,
                LDAPControl **, LDAPControl **);
extern int ldapssl_install_gethostbyaddr(LDAP *ld, const char *skip);

extern int __door_getconf(char **buffer, int *buflen,
                ns_ldap_error_t **error, int callnumber);
extern int __ns_ldap_freeUnixCred(UnixCred_t **credp);
extern int SetDoorInfoToUnixCred(char *buffer,
                ns_ldap_error_t **errorp,
                UnixCred_t **cred);

static int openConnection(LDAP **, const char *, const ns_cred_t *,
                int, ns_ldap_error_t **, int, int, ns_conn_user_t *, int);
static void
_DropConnection(ConnectionID cID, int flag, int fini);

static mutex_t sessionPoolLock = DEFAULTMUTEX;

static Connection **sessionPool = NULL;
static int sessionPoolSize = 0;

/*
 * SSF values are for SASL integrity & privacy.
 * JES DS5.2 does not support this feature but DS6 does.
 * The values between 0 and 65535 can work with both server versions.
 */
#define MAX_SASL_SSF    65535
#define MIN_SASL_SSF    0

/* Number of hostnames to allocate memory for */
#define NUMTOMALLOC     32

/*
 * This function get the servers from the lists and returns
 * the first server with the empty lists of server controls and
 * SASL mechanisms. It is invoked if it is not possible to obtain a server
 * from ldap_cachemgr or the local list.
 */
static
ns_ldap_return_code
getFirstFromConfig(ns_server_info_t *ret, ns_ldap_error_t **error)
{
        char                    **servers = NULL;
        ns_ldap_return_code     ret_code;
        char                    errstr[MAXERROR];

        /* get first server from config list unavailable otherwise */
        ret_code = __s_api_getServers(&servers, error);
        if (ret_code != NS_LDAP_SUCCESS) {
                if (servers != NULL) {
                        __s_api_free2dArray(servers);
                }
                return (ret_code);
        }

        if (servers == NULL || servers[0] == NULL) {
                __s_api_free2dArray(servers);
                (void) sprintf(errstr,
                    gettext("No server found in configuration"));
                MKERROR(LOG_ERR, *error, NS_CONFIG_NODEFAULT,
                    strdup(errstr), NS_LDAP_MEMORY);
                return (NS_LDAP_CONFIG);
```

```
 128         }

 130         ret->server = strdup(servers[0]);
 131         if (ret->server == NULL) {
 132                 __s_api_free2dArray(servers);
 133                 return (NS_LDAP_MEMORY);
 134         }

 136         ret->saslMechanisms = NULL;
 137         ret->controls = NULL;

 139         __s_api_free2dArray(servers);

 141         return (NS_LDAP_SUCCESS);
 142 }
______unchanged_portion_omitted_


 465 #ifdef DEBUG
 466 /*
 467  * printCred(): prints the credential structure
 468  */
 469 static void
 470 printCred(FILE *fp, const ns_cred_t *cred)
 471 {
 472         thread_t        t = thr_self();

 474         if (cred == NULL) {
 475                 (void) fprintf(fp, "tid= %d: printCred: cred is NULL\n", t);
 476                 return;
 477         }

 479         (void) fprintf(fp, "tid= %d: AuthType=%d\n", t, cred->auth.type);
 480         (void) fprintf(fp, "tid= %d: TlsType=%d\n", t, cred->auth.tlstype);
 481         (void) fprintf(fp, "tid= %d: SaslMech=%d\n", t, cred->auth.saslmech);
 482         (void) fprintf(fp, "tid= %d: SaslOpt=%d\n", t, cred->auth.saslopt);
 478         (void) fprintf(fp, "tid= %d: AuthType=%d", t, cred->auth.type);
 479         (void) fprintf(fp, "tid= %d: TlsType=%d", t, cred->auth.tlstype);
 480         (void) fprintf(fp, "tid= %d: SaslMech=%d", t, cred->auth.saslmech);
 481         (void) fprintf(fp, "tid= %d: SaslOpt=%d", t, cred->auth.saslopt);
 483         if (cred->hostcertpath)
 484                 (void) fprintf(fp, "tid= %d: hostCertPath=%s\n",
 485                     t, cred->hostcertpath);
 486         if (cred->cred.unix_cred.userID)
 487                 (void) fprintf(fp, "tid= %d: userID=%s\n",
 488                     t, cred->cred.unix_cred.userID);
 489         if (cred->cred.unix_cred.passwd)
 490                 (void) fprintf(fp, "tid= %d: passwd=%s\n",
 491                     t, cred->cred.unix_cred.passwd);
 492 }
______unchanged_portion_omitted_

 675 /*
 676  * Find a connection matching the passed in criteria.  If an open
 677  * connection with that criteria exists use it, otherwise open a
 678  * new connection.
 679  * Success: returns the pointer to the Connection structure
 680  * Failure: returns NULL, error code and message should be in errorp
 681  */

 683 static int
 684 makeConnection(Connection **conp, const char *serverAddr,
 685         const ns_cred_t *auth, ConnectionID *cID, int timeoutSec,
 686         ns_ldap_error_t **errorp, int fail_if_new_pwd_reqd,
 687         int nopasswd_acct_mgmt, int flags, char ***badsrvrs,
 688         ns_conn_user_t *conn_user)
```



```
 689 {
 690         Connection *con = NULL;
 691         ConnectionID id;
 692         char errmsg[MAXERROR];
 693         int rc, exit_rc = NS_LDAP_SUCCESS;
 694         ns_server_info_t sinfo;
 695         char *hReq, *host = NULL;
 696         LDAP *ld = NULL;
 697         int passwd_mgmt = 0;
 698         int totalbad = 0; /* Number of servers contacted unsuccessfully */
 699         short   memerr = 0; /* Variable for tracking memory allocation */
 700         char *serverAddrType = NULL, **bindHost = NULL;


 703         if (conp == NULL || errorp == NULL || auth == NULL)
 704                 return (NS_LDAP_INVALID_PARAM);
 705         if (*errorp)
 706                 __ns_ldap_freeError(errorp);
 704         *errorp = NULL;
 707         *conp = NULL;
 708         (void) memset(&sinfo, 0, sizeof (sinfo));

 710         if ((id = findConnection(flags, serverAddr, auth, &con)) != -1) {
 711                 /* connection found in cache */
 712 #ifdef DEBUG
 713                 (void) fprintf(stderr, "tid= %d: connection found in "
 714                     "cache %d\n", thr_self(), id);
 715                 fflush(stderr);
 716 #endif /* DEBUG */
 717                 *cID = id;
 718                 *conp = con;
 719                 return (NS_LDAP_SUCCESS);
 720         }

 722         if (auth->auth.saslmech == NS_LDAP_SASL_GSSAPI) {
 723                 serverAddrType = NS_CACHE_ADDR_HOSTNAME;
 724                 bindHost = &sinfo.serverFQDN;
 725         } else {
 726                 serverAddrType = NS_CACHE_ADDR_IP;
 727                 bindHost = &sinfo.server;
 728         }

 730         if (serverAddr) {
 731                 if (__s_api_isInitializing()) {
 732                         /*
 733                          * When obtaining the root DSE, connect to the server
 734                          * passed here through the serverAddr parameter
 735                          */
 736                         sinfo.server = strdup(serverAddr);
 737                         if (sinfo.server == NULL)
 738                                 return (NS_LDAP_MEMORY);
 739                         if (strcmp(serverAddrType,
 740                             NS_CACHE_ADDR_HOSTNAME) == 0) {
 741                                 rc = __s_api_ip2hostname(sinfo.server,
 742                                     &sinfo.serverFQDN);
 743                                 if (rc != NS_LDAP_SUCCESS) {
 744                                         (void) snprintf(errmsg,
 745                                             sizeof (errmsg),
 746                                             gettext("The %s address "
 747                                             "can not be resolved into "
 748                                             "a host name. Returning "
 749                                             "the address as it is."),
 750                                             serverAddr);
 751                                         MKERROR(LOG_ERR,
 752                                             *errorp,
 753                                             NS_CONFIG_NOTLOADED,
```

```
754                                         strdup(errmsg),
755                                         NS_LDAP_MEMORY);
756                                 __s_api_free_server_info(&sinfo);
757                                 return (NS_LDAP_INTERNAL);
758                         }
759                 }
760         } else {
761                 /*
762                  * We're given the server address, just use it.
763                  * In case of sasl/GSSAPI, serverAddr would need
764                  * to be a FQDN.  We assume this is the case for now.
765                  *
766                  * Only the server address fields of sinfo structure
767                  * are filled in since these are the only relevant
768                  * data that we have. Other fields of this structure
769                  * (controls, saslMechanisms) are kept to NULL.
770                  */
771                 sinfo.server = strdup(serverAddr);
772                 if (sinfo.server == NULL)  {
773                         return (NS_LDAP_MEMORY);
774                 }
775                 if (auth->auth.saslmech == NS_LDAP_SASL_GSSAPI) {
776                         sinfo.serverFQDN = strdup(serverAddr);
777                         if (sinfo.serverFQDN == NULL) {
778                                 free(sinfo.server);
779                                 return (NS_LDAP_MEMORY);
780                         }
781                 }
782         }
783         rc = openConnection(&ld, *bindHost, auth, timeoutSec, errorp,
784             fail_if_new_pwd_reqd, passwd_mgmt, conn_user, flags);
785         if (rc == NS_LDAP_SUCCESS || rc ==
786             NS_LDAP_SUCCESS_WITH_INFO) {
787                 exit_rc = rc;
788                 goto create_con;
789         } else {
790                 if (auth->auth.saslmech == NS_LDAP_SASL_GSSAPI) {
791                         (void) snprintf(errmsg, sizeof (errmsg),
792                             "%s %s", gettext("makeConnection: "
793                             "failed to open connection using "
794                             "sasl/GSSAPI to"), *bindHost);
795                 } else {
796                         (void) snprintf(errmsg, sizeof (errmsg),
797                             "%s %s", gettext("makeConnection: "
798                             "failed to open connection to"),
799                             *bindHost);
800                 }
801                 syslog(LOG_ERR, "libsldap: %s", errmsg);
802                 __s_api_free_server_info(&sinfo);
803                 return (rc);
804         }
805 }

807 /* No cached connection, create one */
808 for (; ; ) {
809         if (host == NULL)
810                 hReq = NS_CACHE_NEW;
811         else
812                 hReq = NS_CACHE_NEXT;
813         rc = __s_api_requestServer(hReq, host, &sinfo, errorp,
814             serverAddrType);
815         if ((rc != NS_LDAP_SUCCESS) || (sinfo.server == NULL) ||
816             (host && (strcasecmp(host, sinfo.server) == 0))) {
817                 /* Log the error */
818                 if (*errorp) {
819                         (void) snprintf(errmsg, sizeof (errmsg),
```



```
820                             "%s: (%s)", gettext("makeConnection: "
821                             "unable to make LDAP connection, "
822                             "request for a server failed"),
823                                 (*errorp)->message);
824                             syslog(LOG_ERR, "libsldap: %s", errmsg);
825                     }

827                     __s_api_free_server_info(&sinfo);
828                     if (host)
829                             free(host);
830                     return (NS_LDAP_OP_FAILED);
831             }
832             if (host)
833                     free(host);
834             host = strdup(sinfo.server);
835             if (host == NULL) {
836                     __s_api_free_server_info(&sinfo);
837                     return (NS_LDAP_MEMORY);
838             }

840             /* check if server supports password management */
841             passwd_mgmt = __s_api_contain_passwd_control_oid(
842                 sinfo.controls);
843             /* check if server supports password less account mgmt */
844             if (nopasswd_acct_mgmt &&
845                 !__s_api_contain_account_usable_control_oid(
846                 sinfo.controls)) {
847                     syslog(LOG_WARNING, "libsldap: server %s does not "
848                         "provide account information without password",
849                         host);
850                     free(host);
851                     __s_api_free_server_info(&sinfo);
852                     return (NS_LDAP_OP_FAILED);
853             }
854             /* make the connection */
855             rc = openConnection(&ld, *bindHost, auth, timeoutSec, errorp,
856                 fail_if_new_pwd_reqd, passwd_mgmt, conn_user, flags);
857             /* if success, go to create connection structure */
858             if (rc == NS_LDAP_SUCCESS ||
859                 rc == NS_LDAP_SUCCESS_WITH_INFO) {
860                     exit_rc = rc;
861                     break;
862             }

864             /*
865              * If not able to reach the server, inform the ldap
866              * cache manager that the server should be removed
867              * from its server list. Thus, the manager will not
868              * return this server on the next get-server request
869              * and will also reduce the server list refresh TTL,
870              * so that it will find out sooner when the server
871              * is up again.
872              */
873             if (rc == NS_LDAP_INTERNAL && *errorp != NULL) {
874                     if ((*errorp)->status == LDAP_CONNECT_ERROR ||
875                         (*errorp)->status == LDAP_SERVER_DOWN) {
876                             /* Reset memory allocation error */
877                             memerr = 0;
878                             /*
879                              * We contacted a server that we could
880                              * not either authenticate to or contact.
881                              * If it is due to authentication, then
882                              * we need to try the server again. So,
883                              * do not remove the server yet, but
884                              * add it to the bad server list.
885                              * The caller routine will remove
```

```
886                                 * the servers if:
887                                 *      a). A good server is found or
888                                 *      b). All the possible methods
889                                 *          are tried without finding
890                                 *          a good server
891                                 */
892                                if (*badsrvrs == NULL) {
893                                      if (!(*badsrvrs = (char **)malloc
894                                          (sizeof (char *) * NUMTOMALLOC))) {
895                                              memerr = 1;
896                                      }
897                                /* Allocate memory in chunks of NUMTOMALLOC */
898                                } else if ((totalbad % NUMTOMALLOC) ==
899                                     NUMTOMALLOC - 1) {
900                                      char **tmpptr;
901                                      if (!(tmpptr = (char **)realloc(
902                                          *badsrvrs,
903                                          (sizeof (char *) * NUMTOMALLOC *
904                                          ((totalbad/NUMTOMALLOC) + 2)))) {
905                                              memerr = 1;
906                                      } else {
907                                              *badsrvrs = tmpptr;
908                                      }
909                                }
910                                /*
911                                 * Store host only if there were no unsuccessful
912                                 * memory allocations above
913                                 */
914                                if (!memerr &&
915                                    !((*badsrvrs)[totalbad++] = strdup(host))) {
916                                      memerr = 1;
917                                      totalbad--;
918                                }
919                                (*badsrvrs)[totalbad] = NULL;
920                        }
921                }

923                /* else, cleanup and go for the next server */
924                __s_api_free_server_info(&sinfo);

926                /* Return if we had memory allocation errors */
927                if (memerr)
928                        return (NS_LDAP_MEMORY);
929                if (*errorp) {
930                        /*
931                         * If openConnection() failed due to
932                         * password policy, or invalid credential,
933                         * keep *errorp and exit
934                         */
935                        if ((*errorp)->pwd_mgmt.status != NS_PASSWD_GOOD ||
936                            (*errorp)->status == LDAP_INVALID_CREDENTIALS) {
937                                free(host);
938                                return (rc);
939                        } else {
940                                (void) __ns_ldap_freeError(errorp);
941                                *errorp = NULL;
942                        }
943                }
944        }

946 create_con:
947        /* we have created ld, setup con structure */
948        if (host)
949                free(host);
950        if ((con = calloc(1, sizeof (Connection))) == NULL) {
951                __s_api_free_server_info(&sinfo);
```



```
952                /*
953                 * If password control attached in **errorp,
954                 * e.g. rc == NS_LDAP_SUCCESS_WITH_INFO,
955                 * free the error structure
956                 */
957                if (*errorp) {
958                        (void) __ns_ldap_freeError(errorp);
959                        *errorp = NULL;
960                }
961                (void) ldap_unbind(ld);
962                return (NS_LDAP_MEMORY);
963        }

965        con->serverAddr = sinfo.server; /* Store original format */
966        if (sinfo.serverFQDN != NULL) {
967                free(sinfo.serverFQDN);
968                sinfo.serverFQDN = NULL;
969        }
970        con->saslMechanisms = sinfo.saslMechanisms;
971        con->controls = sinfo.controls;

973        con->auth = __ns_ldap_dupAuth(auth);
974        if (con->auth == NULL) {
975                (void) ldap_unbind(ld);
976                __s_api_freeConnection(con);
977                /*
978                 * If password control attached in **errorp,
979                 * e.g. rc == NS_LDAP_SUCCESS_WITH_INFO,
980                 * free the error structure
981                 */
982                if (*errorp) {
983                        (void) __ns_ldap_freeError(errorp);
984                        *errorp = NULL;
985                }
986                return (NS_LDAP_MEMORY);
987        }

989        con->threadID = thr_self();
990        con->pid = getpid();

992        con->ld = ld;
993        /* add MT connection to the MT connection pool */
994        if (conn_user != NULL && conn_user->conn_mt != NULL) {
995                if (__s_api_conn_mt_add(con, conn_user, errorp) ==
996                    NS_LDAP_SUCCESS) {
997                        *conp = con;
998                        return (exit_rc);
999                } else {
1000                       (void) ldap_unbind(ld);
1001                       __s_api_freeConnection(con);
1002                       return ((*errorp)->status);
1003               }
1004       }

1006       /* MT connection not supported or not required case */
1007       if ((id = addConnection(con)) == -1) {
1008               (void) ldap_unbind(ld);
1009               __s_api_freeConnection(con);
1010               /*
1011                * If password control attached in **errorp,
1012                * e.g. rc == NS_LDAP_SUCCESS_WITH_INFO,
1013                * free the error structure
1014                */
1015               if (*errorp) {
1016                       (void) __ns_ldap_freeError(errorp);
1017                       *errorp = NULL;
```

```
1018                 }
1019                 return (NS_LDAP_MEMORY);
1020         }
1021 #ifdef DEBUG
1022         (void) fprintf(stderr, "tid= %d: connection added into "
1023             "cache %d\n", thr_self(), id);
1024         fflush(stderr);
1025 #endif /* DEBUG */
1026         *cID = id;
1027         *conp = con;
1028         return (exit_rc);
1029 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
   60941 Wed Oct 17 22:00:50 2012
new/usr/src/lib/libsldap/common/ns_standalone.c
3285 memory leaks in libsldap
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */
  21 /*
  22  * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  23  * Use is subject to license terms.
  24  * Copyright 2012 Milan Jurik. All rights reserved.
  25  * Copyright 2012 Nexenta Systems, Inc. All rights reserved.
  26  */

  28 #define __STANDALONE_MODULE__

  30 #include <stdio.h>
  31 #include <sys/types.h>
  32 #include <stdlib.h>
  33 #include <libintl.h>
  34 #include <string.h>
  35 #include <ctype.h>

  37 #include <sys/stat.h>
  38 #include <fcntl.h>
  39 #include <unistd.h>
  40 #include <syslog.h>
  41 #include <locale.h>
  42 #include <errno.h>
  43 #include <sys/time.h>

  45 #include <arpa/inet.h>
  46 #include <netdb.h>
  47 #include <strings.h>

  49 #include <thread.h>

  51 #include <nsswitch.h>
  52 #include <nss_dbdefs.h>
  53 #include <nss.h>

  55 #include "ns_cache_door.h"
  56 #include "ns_internal.h"
  57 #include "ns_connmgmt.h"

  59 typedef enum {
  60         INFO_SERVER_JUST_INITED = -1,
  61         INFO_SERVER_UNKNOWN     = 0,
```



```
  62         INFO_SERVER_CONNECTING  = 1,
  63         INFO_SERVER_UP          = 2,
  64         INFO_SERVER_ERROR       = 3,
  65         INFO_SERVER_REMOVED     = 4
  66 } dir_server_status_t;
______unchanged_portion_omitted_

1281 /*
1282  * This function obtains the root DSE from a specified server.
1283  *
1284  * INPUT:
1285  *     server_addr - an adress of a server to be connected to.
1286  *
1287  * OUTPUT:
1288  *     root_dse - a buffer containing the root DSE in the following format:
1289  *          [<attribute name>=]value [DOORLINESEP [<attribute name>=]value ]...]
1290  *        For example: ( here | used as DOORLINESEP for visual purposes)
1291  *          supportedControl=1.1.1.1|supportedSASLmechanisms=EXTERNAL
1292  *        Should be free'ed by the caller.
1293  */
1294 ns_ldap_return_code
1295 __ns_ldap_getRootDSE(const char *server_addr,
1296                 char **root_dse,
1297                 ns_ldap_error_t **errorp,
1298                 int anon_fallback)
1299 {
1300         char                    errmsg[MAXERROR];
1301         ns_ldap_return_code     ret_code;

1303         ConnectionID            sessionId = 0;
1304         Connection              *session = NULL;

1306         struct timeval          tv = {NS_DEFAULT_SEARCH_TIMEOUT, 0};
1307         char                    *attrs[3];
1308         int                     ldap_rc, ldaperrno = 0;
1309         LDAPMessage             *resultMsg = NULL;
1310         void                    **paramVal = NULL;

1312         ns_cred_t               anon;
1313         ns_conn_user_t          *cu = NULL;

1315         if (errorp == NULL) {
1316                 return (NS_LDAP_INVALID_PARAM);
1317         }

1319         *errorp = NULL;

1321         if (!root_dse) {
1322                 return (NS_LDAP_INVALID_PARAM);
1323         }

1325         if (!server_addr) {
1326                 return (NS_LDAP_INVALID_PARAM);
1327         }

1329         __s_api_setInitMode();

1331         cu = __s_api_conn_user_init(NS_CONN_USER_SEARCH, NULL, B_FALSE);
1332         if (cu == NULL) {
1333                 return (NS_LDAP_INTERNAL);
1334         }

1336         /*
1337          * All the credentials will be taken from the current
1338          * libsldap configuration.
1339          */
```

```
1340         if ((ret_code = __s_api_getConnection(server_addr,
1341             NS_LDAP_NEW_CONN,
1342             NULL,
1343             &sessionId,
1344             &session,
1345             errorp,
1346             0,
1347             0,
1348             cu)) != NS_LDAP_SUCCESS) {
1349                 /* Fallback to anonymous mode is disabled. Stop. */
1350                 if (anon_fallback == 0) {
1351                         syslog(LOG_WARNING,
1352                             gettext("libsldap: can not get the root DSE from "
1353                             " the %s server: %s. "
1354                             "Falling back to anonymous disabled.\n"),
1355                             server_addr,
1356                             errorp && *errorp && (*errorp)->message ?
1357                             (*errorp)->message : "");
1358                         if (errorp != NULL && *errorp != NULL) {
1359                                 (void) __ns_ldap_freeError(errorp);
1360                         }
1361                         __s_api_unsetInitMode();
1362                         return (ret_code);
1363                 }

1365                 /*
1366                  * Fallback to anonymous, non-SSL mode for backward
1367                  * compatibility reasons. This mode should only be used when
1368                  * this function (__ns_ldap_getRootDSE) is called from
1369                  * ldap_cachemgr(1M).
1370                  */
1371                 syslog(LOG_WARNING,
1372                     gettext("libsldap: Falling back to anonymous, non-SSL"
1373                     " mode for __ns_ldap_getRootDSE. %s\n"),
1374                     errorp && *errorp && (*errorp)->message ?
1375                     (*errorp)->message : "");

1377                 /* Setup the anon credential for anonymous connection. */
1378                 (void) memset(&anon, 0, sizeof (ns_cred_t));
1379                 anon.auth.type = NS_LDAP_AUTH_NONE;

1381                 if (*errorp != NULL) {
1382                         (void) __ns_ldap_freeError(errorp);
1383                 }
1384                 *errorp = NULL;

1386                 ret_code = __s_api_getConnection(server_addr,
1387                     NS_LDAP_NEW_CONN,
1388                     &anon,
1389                     &sessionId,
1390                     &session,
1391                     errorp,
1392                     0,
1393                     0,
1394                     cu);

1396                 if (ret_code != NS_LDAP_SUCCESS) {
1397                         __s_api_conn_user_free(cu);
1398                         __s_api_unsetInitMode();
1399                         return (ret_code);
1400                 }
1401         }

1403         __s_api_unsetInitMode();

1405         /* get search timeout value */
```



```
1406         (void) __ns_ldap_getParam(NS_LDAP_SEARCH_TIME_P, &paramVal, errorp);
1407         if (paramVal != NULL && *paramVal != NULL) {
1408                 tv.tv_sec = **((int **)paramVal);
1409                 (void) __ns_ldap_freeParam(&paramVal);
1410         }
1411         if (*errorp != NULL) {
1412                 (void) __ns_ldap_freeError(errorp);
1413         }

1415         /* Get root DSE from the server specified by the caller. */
1416         attrs[0] = "supportedControl";
1417         attrs[1] = "supportedsaslmechanisms";
1418         attrs[2] = NULL;
1419         ldap_rc = ldap_search_ext_s(session->ld,
1420             "",
1421             LDAP_SCOPE_BASE,
1422             "(objectclass=*)",
1423             attrs,
1424             0,
1425             NULL,
1426             NULL,
1427             &tv,
1428             0,
1429             &resultMsg);

1431         if (ldap_rc != LDAP_SUCCESS) {
1432                 /*
1433                  * If the root DSE was not found, the server does
1434                  * not comply with the LDAP v3 protocol.
1435                  */
1436                 (void) ldap_get_option(session->ld,
1437                     LDAP_OPT_ERROR_NUMBER,
1438                     &ldaperrno);
1439                 (void) snprintf(errmsg,
1440                     sizeof (errmsg),
1441                     gettext(ldap_err2string(ldaperrno)));
1442                 MKERROR(LOG_ERR,
1443                     *errorp,
1444                     NS_LDAP_OP_FAILED,
1445                     strdup(errmsg),
1446                     NS_LDAP_MEMORY);

1448                 if (resultMsg) {
1449                         (void) ldap_msgfree(resultMsg);
1450                         resultMsg = NULL;
1451                 }

1453                 __s_api_conn_user_free(cu);
1454                 DropConnection(sessionId, NS_LDAP_NEW_CONN);
1455                 return (NS_LDAP_OP_FAILED);
1456         }
1457         __s_api_conn_user_free(cu);

1459         ret_code = convert_to_door_line(session->ld,
1460             resultMsg,
1461             INCLUDE_ATTR_NAMES,
1462             NOT_PROFILE,
1463             root_dse);
1464         if (ret_code == NS_LDAP_NOTFOUND) {
1465                 (void) snprintf(errmsg,
1466                     sizeof (errmsg),
1467                     gettext("No root DSE data "
1468                     "for server %s returned."),
1469                     server_addr);
1470                 MKERROR(LOG_ERR,
1471                     *errorp,
```

```
1472                    NS_LDAP_NOTFOUND,
1473                    strdup(errmsg),
1474                    NS_LDAP_MEMORY);
1475        }

1477        if (resultMsg) {
1478                (void) ldap_msgfree(resultMsg);
1479                resultMsg = NULL;
1480        }

1482        DropConnection(sessionId, NS_LDAP_NEW_CONN);

1484        return (ret_code);
1485 }
```

_____***unchanged_portion_omitted_***