```
**********************************************************
    2080 Thu Jul  9 15:59:30 2015
new/usr/src/lib/libresolv/Makefile
1926 libresolv evades compiler warnings
**********************************************************
```

```
   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 2015 Gary Mills
  23 # Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  24 # Use is subject to license terms.
  25 #

  27 LIBRARY= libresolv.a
  28 VERS= .1

  30 PICS=   pics/res_comp.o pics/res_debug.o pics/res_init.o \
  31         pics/res_mkquery.o pics/res_query.o pics/res_send.o \
  32         pics/res_gethost.o pics/res_sethost.o

  34 pics/%.o: %.c
  35         $(COMPILE.c) -o $@ $<
  36         $(POST_PROCESS_O)

  38 OBJECTS= \
  39 res_gethost.o   res_comp.o      res_debug.o     res_init.o    res_mkquery.o  \
  40 res_query.o     res_send.o      res_sethost.o

  42 # include library definitions
  43 include ../Makefile.lib

  45 # install this library in the root filesystem
  46 include ../Makefile.rootfs

  48 SRCDIR =        .

  50 C99MODE=        $(C99_DISABLE)

  52 # We really want to say this:
  53 #       CPPFLAGS += -DDEBUG -DSYSV -D_REENTRANT -I. -I../common/inc
  54 # but some system header files are replaced by local versions
  55 # so we must put -I. ahead of the default include directories:
  56 CPPFLAGS = -I. -I../common/inc $(CPPFLAGS.master) -DDEBUG -DSYSV -D_REENTRANT
  57 LDLIBS += -lsocket -lnsl -lc
  58 CFLAGS += $(CCVERBOSE)

  58 CERRWARN += -_gcc=-Wno-implicit-function-declaration
  60 CERRWARN += -_gcc=-Wno-parentheses
```



```
  60 CERRWARN += -_gcc=-Wno-unused-variable
  61 CERRWARN += -_gcc=-Wno-uninitialized
  62 CERRWARN += -_gcc=-Wno-implicit-int
  63 CERRWARN += -_gcc=-Wno-extra

  63 ROOTDYNLIBS=    $(DYNLIB:%=$(ROOTLIBDIR)/%)

  65 .KEEP_STATE:

  67 LIBS = $(DYNLIB)

  69 all: $(LIBS)

  71 install: all $(ROOTDYNLIBS)

  73 lint: lintcheck

  75 # include library targets
  76 include ../Makefile.targ
```

****
1309 Thu Jul 9 15:59:31 2015
new/usr/src/lib/libresolv/crossl.h
1926 libresolv evades compiler warnings
****

```
/*
 * This file and its contents are supplied under the terms of the
 * Common Development and Distribution License ("CDDL"), version 1.0.
 * You may only use this file in accordance with the terms of version
 * 1.0 of the CDDL.
 *
 * A full copy of the text of the CDDL should have accompanied this
 * source.  A copy of the CDDL is also available via the Internet at
 * http://www.illumos.org/license/CDDL.
 */

/*
 * Copyright 2015 Gary Mills
 */

#ifndef _CROSSL_H
#define _CROSSL_H

/*
 * Definitions needed for cross-linkages between source files
 */

#ifdef __cplusplus
extern "C" {
#endif

extern int
dn_comp(u_char *, u_char *, int, u_char **, u_char **);
extern int
dn_expand(u_char *, u_char *, u_char *, u_char *, int);
extern int
dn_skipname(u_char *, u_char *);

extern int
res_init(void);
extern int
res_mkquery(int, char *, int, int, char *, int, struct rrec *, char *, int);
extern int
res_query(char *, int, int, u_char *, int);
extern int
res_querydomain(char *, char *, int, int, u_char *, int);
extern int
res_search(char *, int, int, u_char *, int);
extern int
res_send(char *, int, char *, int);

extern void
putlong(u_long, u_char *);
extern void
putshort(u_short, u_char *);
extern void
p_query(char *);
extern void
_res_close();


#ifdef __cplusplus
}
#endif

#endif /* _CROSSL_H */
```

```
***********************************************************
   10957 Thu Jul  9 15:59:31 2015
new/usr/src/lib/libresolv/res_debug.c
1926 libresolv evades compiler warnings
***********************************************************
```

```
/*
 * 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 2015 Gary Mills
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

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

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <arpa/nameser.h>
#include <resolv.h>
#include "crossl.h"

extern char *p_cdname(), *p_rr(), *p_type(), *p_class(), *p_time();
extern char *inet_ntoa();
void fp_query(char *msg, FILE *file);

char *_res_opcodes[] = {
        "QUERY",
        "IQUERY",
        "CQUERYM",
        "CQUERYU",
```



```
        "4",
        "5",
        "6",
        "7",
        "8",
        "UPDATEA",
        "UPDATED",
        "UPDATEDA",
        "UPDATEM",
        "UPDATEMA",
        "ZONEINIT",
        "ZONEREF",
};
_____unchanged_portion_omitted_

char *
p_cdname(cp, msg, file)
        char *cp, *msg;
        FILE *file;
{
        char name[MAXDNAME];
        int n;

        if ((n = dn_expand((u_char *)msg, (u_char *)(msg + 512), (u_char *)cp,
            (u_char *)name, sizeof (name))) < 0)
        if ((n = dn_expand(msg, msg + 512, cp, name, sizeof (name))) < 0)
                return (NULL);
        if (name[0] == '\0') {
                name[0] = '.';
                name[1] = '\0';
        }
        fputs(name, file);
        return (cp + n);
}
_____unchanged_portion_omitted_
```

```
**********************************************************
    9953 Thu Jul  9 15:59:31 2015
new/usr/src/lib/libresolv/res_gethost.c
1926 libresolv evades compiler warnings
**********************************************************
```

```
/*
 * Copyright 2015 Gary Mills
 * Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * Copyright (c) 1985, 1988 Regents of the University of California.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms are permitted
 * provided that this notice is preserved and that due credit is given
 * to the University of California at Berkeley. The name of the University
 * may not be used to endorse or promote products derived from this
 * software without specific prior written permission. This software
 * is provided ''as is'' without express or implied warranty.
 *
 */

#include <sys/param.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <ctype.h>
#include <netdb.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <arpa/inet.h>
#include <arpa/nameser.h>
#include <resolv.h>
#include <syslog.h>
#include "crossl.h"

/*
 * When the name service switch calls libresolv, it doesn't want fallback
 * to /etc/hosts, so we provide a method to turn it off.
 */
static int      no_hosts_fallback = 0;

void
__res_set_no_hosts_fallback(void) {
        no_hosts_fallback = 1;
}
```

*unchanged_portion_omitted_*

```
int h_errno;

static struct hostent *
getanswer(answer, anslen, iquery)
        querybuf *answer;
        int anslen;
        int iquery;
{
        register HEADER *hp;
        register u_char *cp;
        register int n;
        u_char *eom;
        char *bp, **ap;
        int type, class, buflen, ancount, qdcount;
        int haveanswer, getclass = C_ANY;
```



```
        char **hap;

        eom = answer->buf + anslen;
        /*
         * find first satisfactory answer
         */
        hp = &answer->hdr;
        ancount = ntohs(hp->ancount);
        qdcount = ntohs(hp->qdcount);
        bp = hostbuf;
        buflen = sizeof (hostbuf);
        cp = answer->buf + sizeof (HEADER);
        if (qdcount) {
                if (iquery) {
                        if ((n = dn_expand(answer->buf, eom,
                            cp, (u_char *)bp, buflen)) < 0) {
                        if ((n = dn_expand((char *)answer->buf, eom,
                                                cp, bp, buflen)) < 0) {
                                h_errno = NO_RECOVERY;
                                return ((struct hostent *) NULL);
                        }
                        cp += n + QFIXEDSZ;
                        host.h_name = bp;
                        n = strlen(bp) + 1;
                        bp += n;
                        buflen -= n;
                } else
                        cp += dn_skipname(cp, eom) + QFIXEDSZ;
                while (--qdcount > 0)
                        cp += dn_skipname(cp, eom) + QFIXEDSZ;
        } else if (iquery) {
                if (hp->aa)
                        h_errno = HOST_NOT_FOUND;
                else
                        h_errno = TRY_AGAIN;
                return ((struct hostent *) NULL);
        }
        ap = host_aliases;
        host.h_aliases = host_aliases;
        hap = h_addr_ptrs;
#if BSD >= 43 || defined(h_addr)        /* new-style hostent structure */
        host.h_addr_list = h_addr_ptrs;
#endif
        haveanswer = 0;
        while (--ancount >= 0 && cp < eom && haveanswer < MAXADDRS) {
                if ((n = dn_expand(answer->buf, eom,
                    cp, (u_char *)bp, buflen)) < 0)
                if ((n = dn_expand((char *)answer->buf, eom,
                                                cp, bp, buflen)) < 0)
                        break;
                cp += n;
                type = _getshort(cp);
                cp += sizeof (u_short);
                class = _getshort(cp);
                cp += sizeof (u_short) + sizeof (u_long);
                n = _getshort(cp);
                cp += sizeof (u_short);
                if (type == T_CNAME) {
                        cp += n;
                        if (ap >= &host_aliases[MAXALIASES-1])
                                continue;
                        *ap++ = bp;
                        n = strlen(bp) + 1;
                        bp += n;
                        buflen -= n;
                        continue;
```

```
157                 }
158                 if (iquery && type == T_PTR) {
159                         if ((n = dn_expand(answer->buf, eom,
160                             cp, (u_char *)bp, buflen)) < 0) {
156                         if ((n = dn_expand((char *)answer->buf, eom,
157                                         cp, bp, buflen)) < 0) {
161                                 cp += n;
162                                 continue;
163                         }
164                         cp += n;
165                         host.h_name = bp;
166                         return (&host);
167                 }
168                 if (iquery || type != T_A) {
169 #ifdef DEBUG
170                         if (_res.options & RES_DEBUG)
171                                 printf("unexpected answer type %d, size %d\n",
172                                         type, n);
173 #endif
174                         cp += n;
175                         continue;
176                 }
177                 if (haveanswer) {
178                         if (n != host.h_length) {
179                                 cp += n;
180                                 continue;
181                         }
182                         if (class != getclass) {
183                                 cp += n;
184                                 continue;
185                         }
186                 } else {
187                         host.h_length = n;
188                         getclass = class;
189                         host.h_addrtype = (class == C_IN) ? AF_INET : AF_UNSPEC;
190                         if (!iquery) {
191                                 host.h_name = bp;
192                                 bp += strlen(bp) + 1;
193                         }
194                 }

196                 bp += sizeof (align) - ((u_long)bp % sizeof (align));

198                 if (bp + n >= &hostbuf[sizeof (hostbuf)]) {
199 #ifdef DEBUG
200                         if (_res.options & RES_DEBUG)
201                                 printf("size (%d) too big\n", n);
202 #endif
203                         break;
204                 }
205 #ifdef SYSV
206                 memcpy((void *)(*hap++ = bp), (void *)cp, n);
207 #else
208                 bcopy(cp, *hap++ = bp, n);
209 #endif
210                 bp += n;
211                 cp += n;
212                 haveanswer++;
213         }
214         if (haveanswer) {
215                 *ap = NULL;
216 #if BSD >= 43 || defined(h_addr)        /* new-style hostent structure */
217                 *hap = NULL;
218 #else
219                 host.h_addr = h_addr_ptrs[0];
220 #endif
```



```
221                 return (&host);
222         } else {
223                 h_errno = TRY_AGAIN;
224                 return ((struct hostent *) NULL);
225         }
226 }

228 static struct hostent *_gethtbyname();

230 struct hostent *
231 res_gethostbyname(name)
232         char *name;
233 {
234         querybuf buf;
235         register char *cp;
236         int n;
234         struct hostent *hp, *gethostdomain();

238         /*
239          * disallow names consisting only of digits/dots, unless
240          * they end in a dot.
241          */
242         if (isdigit(name[0]))
243                 for (cp = name; /*EMPTY*/; ++cp) {
244                         if (!*cp) {
245                                 if (*--cp == '.')
246                                         break;
247                                 h_errno = HOST_NOT_FOUND;
248                                 return ((struct hostent *) NULL);
249                         }
250                         if (!isdigit(*cp) && *cp != '.')
251                                 break;
252                 }

254         if ((n = res_search(name, C_IN, T_A, buf.buf, sizeof (buf))) < 0) {
255 #ifdef DEBUG
256                 if (_res.options & RES_DEBUG)
257                         printf("res_search failed\n");
258 #endif
259                 if (errno == ECONNREFUSED)
260                         return (_gethtbyname(name));
261                 else
262                         return ((struct hostent *) NULL);
263         }
264         return (getanswer(&buf, n, 0));
265 }

267 static struct hostent *_gethtbyaddr();

269 static struct hostent *
270 _getrhbyaddr(addr, len, type)
271         char *addr;
272         int len, type;
273 {
274         int n;
275         querybuf buf;
276         register struct hostent *hp;
277         char qbuf[MAXDNAME];

279         if (type != AF_INET)
280                 return ((struct hostent *) NULL);
281         (void) sprintf(qbuf, "%d.%d.%d.%d.in-addr.arpa",
282                 ((unsigned)addr[3] & 0xff),
283                 ((unsigned)addr[2] & 0xff),
284                 ((unsigned)addr[1] & 0xff),
285                 ((unsigned)addr[0] & 0xff));
```

```
286         n = res_query(qbuf, C_IN, T_PTR, (u_char *)&buf, sizeof (buf));
284         n = res_query(qbuf, C_IN, T_PTR, (char *)&buf, sizeof (buf));
287         if (n < 0) {
288 #ifdef DEBUG
289                 if (_res.options & RES_DEBUG)
290                         printf("res_query failed\n");
291 #endif
292                 if (errno == ECONNREFUSED)
293                         return (_gethtbyaddr(addr, len, type));
294                 return ((struct hostent *) NULL);
295         }
296         hp = getanswer(&buf, n, 1);
297         if (hp == NULL)
298                 return ((struct hostent *) NULL);
299         hp->h_addrtype = type;
300         hp->h_length = len;
301         h_addr_ptrs[0] = (char *)&host_addr;
302         h_addr_ptrs[1] = (char *)0;
303         host_addr = *(struct in_addr *)addr;
304         return (hp);
305 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7938 Thu Jul  9 15:59:31 2015
new/usr/src/lib/libresolv/res_init.c
1926 libresolv evades compiler warnings
**********************************************************
```

```
/*
 * 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 2015 Gary Mills
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

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

#include <sys/types.h>
#include <sys/sockio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <stropts.h>
#include <arpa/nameser.h>
#include <resolv.h>

#include <netinet/in.h>
#include <net/if.h>
#include <netinet/if_ether.h>
#include <arpa/inet.h>

/*
 * Undocumented external function in libnsl
 */
extern int
getdomainname(char *, int);

#define MAXIFS  256

/*
 * Resolver state default settings
 */

struct state _res = {
        RES_TIMEOUT,            /* retransmition time interval */
        4,                              /* number of times to retransmit */
        RES_DEFAULT,            /* options flags */
        1,                              /* number of name servers */
};

/*
 * Set up default settings.  If the configuration file exist, the values
 * there will have precedence.  Otherwise, the server address is set to
 * INADDR_LOOPBACK and the default domain name comes from the gethostname().
 * BUT if the NIS/RPC domain name is set, that is used if all else fails.
 *
 * The configuration file should only be used if you want to redefine your
 * domain or run without a server on your machine.
 *
 * Note the user can always override then domain name with the environment
 * variable LOCALDOMAIN which has absolute priority.
 *
 *
 * Return 0 if completes successfully, -1 on error
 */
int
res_init(void)
{
        register FILE *fp;
        register char *cp, **pp;
        register int n;
        char buf[BUFSIZ];
#ifdef SYSV
        extern char *strchr();
#else
        extern char *index();
#endif
        extern char *strcpy(), *strncpy();
        extern char *getenv();
        int nserv = 0;    /* number of nameserver records read from file */
        int haveenv = 0;
        int havesearch = 0;

        _res.nsaddr.sin_addr.s_addr =  htonl(INADDR_LOOPBACK); /* INADDR_ANY */
        _res.nsaddr.sin_family = AF_INET;
        _res.nsaddr.sin_port = htons(NAMESERVER_PORT);
        _res.nscount = 1;

#ifdef SIOCGIFNUM
        {       int numifs, s, n, int_up;
                struct ifconf ifc;
                register struct ifreq *ifrp;
                struct ifreq ifr;
                unsigned bufsize;
                unsigned int flags;
                char *buf;
                extern void *malloc();

                if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
```

```
118                         perror("socket");
119                         return (-1);
120                 }
121                 if (ioctl(s, SIOCGIFNUM, (char *)&numifs) < 0) {
122                         numifs = MAXIFS;
123                 }
124                 bufsize = numifs * sizeof (struct ifreq);
125                 buf = (char *)malloc(bufsize);
126                 if (buf == NULL) {
127                         perror("out of memory");
128                         (void) close(s);
127                         close(s);
129                         return (-1);
130                 }
131                 ifc.ifc_len = bufsize;
132                 ifc.ifc_buf = buf;
133                 if (ioctl(s, SIOCGIFCONF, (char *)&ifc) < 0) {
134                         perror("ifconfig: SIOCGIFCONF");
135                         (void) close(s);
134                         close(s);
136                         free(buf);
137                         return (-1);
138                 }

140                 int_up = 0;
141                 ifrp = ifc.ifc_req;
142                 for (n = ifc.ifc_len / sizeof (struct ifreq); n > 0;
143                                                         n--, ifrp++) {
144                         (void) memset((void *) &ifr, 0, sizeof (ifr));
143                         memset((void *) &ifr, 0, sizeof (ifr));
145                         strncpy(ifr.ifr_name, ifrp->ifr_name,
146                                                 sizeof (ifr.ifr_name));
147                         if (ioctl(s, SIOCGIFFLAGS, (char *)&ifr) < 0) {
148                                 perror("SIOCGIFFLAGS");
149                                 (void) close(s);
148                                 close(s);
150                                 free(buf);
151                                 return (-1);
152                         }
153                         flags = ifr.ifr_flags;
154                         /* we are looking for a non-loopback interface */
155                         if ((flags & IFF_UP) && ((flags & IFF_LOOPBACK) == 0))
156                                 int_up = 1;
157                 }
158                 (void) close(s);
157                 close(s);
159                 free(buf);
160                 if (int_up == 0) /* all the non-LOOPBACK interfaces are DOWN */
161                         return (-1);
162         }
163 #endif /* SIOCGIFNUM */


166         /*
167          * for the benefit of hidden NIS domains, we use the same procedure
168          * as sendmail: convert leading + to dot, then drop to first dot
169          */
170         (void) getdomainname(buf, BUFSIZ);
169         getdomainname(buf, BUFSIZ);
171         if (buf[0] == '+')
172                 buf[0] = '.';
173 #ifdef SYSV
174         cp = strchr(buf, (int)'.');
175 #else
176         cp = index(buf, '.');
177 #endif
```



```
178         if (cp == NULL)
179                 strcpy(_res.defdname, buf);
180         else
181                 strcpy(_res.defdname, cp+1);

183         /* Allow user to override the local domain definition */
184         if ((cp = getenv("LOCALDOMAIN")) != NULL) {
185         (void) strncpy(_res.defdname, cp, sizeof (_res.defdname));
186         haveenv++;
187         }

189         if ((fp = fopen(_PATH_RESCONF, "r")) != NULL) {
190         /* read the config file */
191         while (fgets(buf, sizeof (buf), fp) != NULL) {
192             /* read default domain name */
193             if (!strncmp(buf, "domain", sizeof ("domain") - 1)) {
194                 if (haveenv)    /* skip if have from environ */
195                                 continue;
196                 cp = buf + sizeof ("domain") - 1;
197                 while (*cp == ' ' || *cp == '\t')
198                     cp++;
199                 if ((*cp == '\0') || (*cp == '\n'))
200                     continue;
201                 (void) strncpy(_res.defdname, cp, sizeof (_res.defdname) - 1);
202 #ifdef SYSV
203                 if ((cp = strchr(_res.defdname, (int)'\n')) != NULL)
204 #else
205                 if ((cp = index(_res.defdname, '\n')) != NULL)
206 #endif
207                     *cp = '\0';
208                 havesearch = 0;
209                 continue;
210             }
211             /* set search list */
212             if (!strncmp(buf, "search", sizeof ("search") - 1)) {
213                 if (haveenv)    /* skip if have from environ */
214                     continue;
215                 cp = buf + sizeof ("search") - 1;
216                 while (*cp == ' ' || *cp == '\t')
217                     cp++;
218                 if ((*cp == '\0') || (*cp == '\n'))
219                     continue;
220                 (void) strncpy(_res.defdname, cp, sizeof (_res.defdname) - 1);
221 #ifdef SYSV
222                 if ((cp = strchr(_res.defdname, (int)'\n')) != NULL)
223 #else
224                 if ((cp = index(_res.defdname, '\n')) != NULL)
225 #endif
226                     *cp = '\0';
227                 /*
228                  * Set search list to be blank-separated strings
229                  * on rest of line.
230                  */
231                 cp = _res.defdname;
232                 pp = _res.dnsrch;
233                 *pp++ = cp;
234                 for (n = 0; *cp && pp < _res.dnsrch + MAXDNSRCH; cp++) {
235                     if (*cp == ' ' || *cp == '\t') {
236                         *cp = 0;
237                         n = 1;
238                     } else if (n) {
239                         *pp++ = cp;
240                         n = 0;
241                     }
242                 }
243                 /* null terminate last domain if there are excess */
```

```
244                 while (*cp != '\0' && *cp != ' ' && *cp != '\t')
245                     cp++;
246                 *cp = '\0';
247                 *pp++ = 0;
248                 havesearch = 1;
249                 continue;
250             }
251             /* read nameservers to query */
252             if (!strncmp(buf, "nameserver", sizeof ("nameserver") - 1) &&
253                 (nserv < MAXNS)) {
254                 cp = buf + sizeof ("nameserver") - 1;
255                 while (*cp == ' ' || *cp == '\t')
256                     cp++;
257                 if ((*cp == '\0') || (*cp == '\n'))
258                     continue;
259                 if ((_res.nsaddr_list[nserv].sin_addr.s_addr =
260                                 inet_addr(cp)) == (unsigned) -1) {
261                     _res.nsaddr_list[n].sin_addr.s_addr = INADDR_ANY;
262                     continue;
263                 }
264                 _res.nsaddr_list[nserv].sin_family = AF_INET;
265                 _res.nsaddr_list[nserv].sin_port = htons(NAMESERVER_PORT);
266                 nserv++;
267                 continue;
268             }
269         }
270         if (nserv > 1)
271             _res.nscount = nserv;
272         (void) fclose(fp);
273         }
274         if (_res.defdname[0] == 0) {
275         if (gethostname(buf, sizeof (_res.defdname)) == 0 &&
276 #ifdef SYSV
277             (cp = strchr(buf, (int)'.')))
278 #else
279             (cp = index(buf, '.')))
280 #endif
281                 (void) strcpy(_res.defdname, cp + 1);
282         }

284         /* find components of local domain that might be searched */
285         if (havesearch == 0) {
286         pp = _res.dnsrch;
287         *pp++ = _res.defdname;
288         for (cp = _res.defdname, n = 0; *cp; cp++)
289             if (*cp == '.')
290                 n++;
291         cp = _res.defdname;
292         for (; n >= LOCALDOMAINPARTS && pp < _res.dnsrch + MAXDFLSRCH; n--) {
293 #ifdef SYSV
294             cp = strchr(cp, (int)'.');
295 #else
296             cp = index(cp, '.');
297 #endif
298             *pp++ = ++cp;
299         }
300         *pp++ = 0;
301         }
302         _res.options |= RES_INIT;
303         return (0);
304 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    7788 Thu Jul  9 15:59:31 2015
new/usr/src/lib/libresolv/res_mkquery.c
1926 libresolv evades compiler warnings
**********************************************************
```

```
/*
 * 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 2015 Gary Mills
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

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

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include "crossl.h"

/*
 * Kludge to time out quickly if there is no /etc/resolv.conf
 * and a TCP connection to the local DNS server fails.
 *
 * Moved function from res_send.c to res_mkquery.c.  This
```



```
 * solves a long timeout problem with nslookup.
 *
 * __areweinnamed is needed because there is a possibility that the
 * user might do bad things to resolv.conf and cause in.named to call
 * _confcheck and deadlock the server.
 */

int __areweinnamed()
{
        return (0);
}

static int _confcheck()
{
        int ns;
        struct stat rc_stat;
        struct sockaddr_in ns_sin;


        /* First, we check to see if /etc/resolv.conf exists.
         * If it doesn't, then localhost is mostlikely to be
         * the nameserver.
         */
        if (stat(_PATH_RESCONF, &rc_stat) == -1 && errno == ENOENT) {

                /* Next, we check to see if _res.nsaddr is set to loopback.
                 * If it isn't, it has been altered by the application
                 * explicitly and we then want to bail with success.
                 */
                if (__areweinnamed())
                        return (0);

                if (_res.nsaddr.sin_addr.S_un.S_addr == htonl(INADDR_LOOPBACK))

                        /* Lastly, we try to connect to the TCP port of the
                         * nameserver.  If this fails, then we know that
                         * DNS is misconfigured and we can quickly exit.
                         */
                        ns = socket(AF_INET, SOCK_STREAM, 0);
                        IN_SET_LOOPBACK_ADDR(&ns_sin);
                        ns_sin.sin_port = htons(NAMESERVER_PORT);
                        if (connect(ns, (struct sockaddr *) &ns_sin,
                                    sizeof ns_sin) == -1) {
                                (void) close(ns);
                                close(ns);
                                return(-1);
                        }
                        else {
                                (void) close(ns);
                                close(ns);
                                return(0);
                        }
                }

                return(0);
        }

        return (0);
}

/*
 * Form all types of queries.
 * Returns the size of the result or -1.
 */
int
res_mkquery(op, dname, class, type, data, datalen, newrr, buf, buflen)
```

```
124         int op;                 /* opcode of query */
125         char *dname;            /* domain name */
126         int class, type;        /* class and type of query */
127         char *data;             /* resource record data */
128         int datalen;            /* length of data */
129         struct rrec *newrr;     /* new rr for modify or append */
130         char *buf;              /* buffer to put query */
131         int buflen;             /* size of buffer */
132 {
133         register HEADER *hp;
134         register u_char *cp;
131         register char *cp;
135         register int n;
136         u_char *dnptrs[10], **dpp, **lastdnptr;
133         char *dnptrs[10], **dpp, **lastdnptr;

138 #ifdef DEBUG
139         if (_res.options & RES_DEBUG)
140                 printf("res_mkquery(%d, %s, %d, %d)\n", op, dname, class, type);
141 #endif /* DEBUG */
142
143         /*
144          * Check to see if we can bailout quickly.
145          * Also rerun res_init if we failed in the past.
146          */

148         if ((_res.options & RES_INIT) == 0 && res_init() == -1) {
149                 h_errno = NO_RECOVERY;
150                 return(-1);
151         }
152
153         if (_confcheck() == -1) {
154                 _res.options &= ~RES_INIT;
155                 h_errno = NO_RECOVERY;
156                 return(-1);
157         }

159         /*
160          * Initialize header fields.
161          */
162         if ((buf == NULL) || (buflen < sizeof (HEADER)))
163                 return (-1);
164 #ifdef SYSV
165         (void) memset(buf, 0, sizeof (HEADER));
162         memset(buf, 0, sizeof (HEADER));
166 #else
167         bzero(buf, sizeof (HEADER));
168 #endif
169         hp = (HEADER *) buf;
170         hp->id = htons(++_res.id);
171         hp->opcode = op;
172         hp->pr = (_res.options & RES_PRIMARY) != 0;
173         hp->rd = (_res.options & RES_RECURSE) != 0;
174         hp->rcode = NOERROR;
175         cp = (u_char *)(buf + sizeof (HEADER));
172         cp = buf + sizeof (HEADER);
176         buflen -= sizeof (HEADER);
177         dpp = dnptrs;
178         *dpp++ = (u_char *)buf;
175         *dpp++ = buf;
179         *dpp++ = NULL;
180         lastdnptr = dnptrs + sizeof (dnptrs) / sizeof (dnptrs[0]);
181         /*
182          * perform opcode specific processing
183          */
184         switch (op) {
```



```
185         case QUERY:
186                 if ((buflen -= QFIXEDSZ) < 0)
187                         return (-1);
188                 if ((n = dn_comp((u_char *)dname, cp, buflen,
189                     dnptrs, lastdnptr)) < 0)
185                 if ((n = dn_comp(dname, cp, buflen, dnptrs, lastdnptr)) < 0)
190                         return (-1);
191                 cp += n;
192                 buflen -= n;
193                 putshort(type, cp);
194                 cp += sizeof (u_short);
195                 putshort(class, cp);
196                 cp += sizeof (u_short);
197                 hp->qdcount = htons(1);
198                 if (op == QUERY || data == NULL)
199                         break;
200                 /*
201                  * Make an additional record for completion domain.
202                  */
203                 buflen -= RRFIXEDSZ;
204                 if ((n = dn_comp((u_char *)data, cp, buflen,
205                     dnptrs, lastdnptr)) < 0)
200                 if ((n = dn_comp(data, cp, buflen, dnptrs, lastdnptr)) < 0)
206                         return (-1);
207                 cp += n;
208                 buflen -= n;
209                 putshort(T_NULL, cp);
210                 cp += sizeof (u_short);
211                 putshort(class, cp);
212                 cp += sizeof (u_short);
213                 putlong(0, cp);
214                 cp += sizeof (u_long);
215                 putshort(0, cp);
216                 cp += sizeof (u_short);
217                 hp->arcount = htons(1);
218                 break;

220         case IQUERY:
221                 /*
222                  * Initialize answer section
223                  */
224                 if (buflen < 1 + RRFIXEDSZ + datalen)
225                         return (-1);
226                 *cp++ = '\0';   /* no domain name */
227                 putshort(type, cp);
228                 cp += sizeof (u_short);
229                 putshort(class, cp);
230                 cp += sizeof (u_short);
231                 putlong(0, cp);
232                 cp += sizeof (u_long);
233                 putshort(datalen, cp);
234                 cp += sizeof (u_short);
235                 if (datalen) {
236 #ifdef SYSV
237                         (void) memcpy((void *)cp, (void *)data, datalen);
232                         memcpy((void *)cp, (void *)data, datalen);
238 #else
239                         bcopy(data, cp, datalen);
240 #endif
241                         cp += datalen;
242                 }
243                 hp->ancount = htons(1);
244                 break;

246 #ifdef ALLOW_UPDATES
247         /*
```

```
248          * For UPDATEM/UPDATEMA, do UPDATED/UPDATEDA followed by UPDATEA
249          * (Record to be modified is followed by its replacement in msg.)
250          */
251         case UPDATEM:
252         case UPDATEMA:

254         case UPDATED:
255                 /*
256                  * The res code for UPDATED and UPDATEDA is the same; user
257                  * calls them differently: specifies data for UPDATED; server
258                  * ignores data if specified for UPDATEDA.
259                  */
260         case UPDATEDA:
261                 buflen -= RRFIXEDSZ + datalen;
262                 if ((n = dn_comp((u_char *)dname, cp, buflen,
263                     dnptrs, lastdnptr)) < 0)
257                 if ((n = dn_comp(dname, cp, buflen, dnptrs, lastdnptr)) < 0)
264                         return (-1);
265                 cp += n;
266                 putshort(type, cp);
267                 cp += sizeof (u_short);
268                 putshort(class, cp);
269                 cp += sizeof (u_short);
270                 putlong(0, cp);
271                 cp += sizeof (u_long);
272                 putshort(datalen, cp);
273                 cp += sizeof (u_short);
274                 if (datalen) {
275 #ifdef SYSV
276                         memcpy((void *)cp, (void *)data, datalen);
277 #else
278                         bcopy(data, cp, datalen);
279 #endif
280                         cp += datalen;
281                 }
282                 if ((op == UPDATED) || (op == UPDATEDA)) {
283                         hp->ancount = htons(0);
284                         break;
285                 }
286                 /* Else UPDATEM/UPDATEMA, so drop into code for UPDATEA */

288         case UPDATEA:   /* Add new resource record */
289                 buflen -= RRFIXEDSZ + datalen;
290                 if ((n = dn_comp((u_char *)dname, cp, buflen,
291                     dnptrs, lastdnptr)) < 0)
284                 if ((n = dn_comp(dname, cp, buflen, dnptrs, lastdnptr)) < 0)
292                         return (-1);
293                 cp += n;
294                 putshort(newrr->r_type, cp);
295                 cp += sizeof (u_short);
296                 putshort(newrr->r_class, cp);
297                 cp += sizeof (u_short);
298                 putlong(0, cp);
299                 cp += sizeof (u_long);
300                 putshort(newrr->r_size, cp);
301                 cp += sizeof (u_short);
302                 if (newrr->r_size) {
303 #ifdef SYSV
304                         memcpy((void *)cp, newrr->r_data, newrr->r_size);
305 #else
306                         bcopy(newrr->r_data, cp, newrr->r_size);
307 #endif
308                         cp += newrr->r_size;
309                 }
310                 hp->ancount = htons(0);
311                 break;
```



```
313 #endif /* ALLOW_UPDATES */
314         }
315         return ((char *)cp - buf);
308         return (cp - buf);
316 }
```

_____*unchanged_portion_omitted_*

```
**********************************************************
    8066 Thu Jul  9 15:59:31 2015
new/usr/src/lib/libresolv/res_query.c
1926 libresolv evades compiler warnings
**********************************************************
```

```
/*
 * 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 2015 Gary Mills
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

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

#include <sys/param.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <ctype.h>
#include <netdb.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <arpa/inet.h>
#include <arpa/nameser.h>
#include <resolv.h>
#include "crossl.h"

#if PACKETSZ > 1024
#define MAXPACKET       PACKETSZ
#else
#define MAXPACKET       1024
#endif
```



```
int h_errno;

/*
 * Formulate a normal query, send, and await answer.
 * Returned answer is placed in supplied buffer "answer".
 * Perform preliminary check of answer, returning success only
 * if no error is indicated and the answer count is nonzero.
 * Return the size of the response on success, -1 on error.
 * Error number is left in h_errno.
 * Caller must parse answer and determine whether it answers the question.
 */
int
res_query(name, class, type, answer, anslen)
        char *name;             /* domain name */
        int class, type;        /* class and type of query */
        u_char *answer;         /* buffer to put answer */
        int anslen;             /* size of answer buffer */
{
        char buf[MAXPACKET];
        HEADER *hp;
        int n;

        if ((_res.options & RES_INIT) == 0 && res_init() == -1)
                return (-1);
#ifdef DEBUG
        if (_res.options & RES_DEBUG)
                printf("res_query(%s, %d, %d)\n", name, class, type);
#endif
        n = res_mkquery(QUERY, name, class, type, (char *)NULL, 0, NULL,
            buf, sizeof (buf));

        if (n <= 0) {
#ifdef DEBUG
                if (_res.options & RES_DEBUG)
                        printf("res_query: mkquery failed\n");
#endif
                h_errno = NO_RECOVERY;
                return (n);
        }
        n = res_send(buf, n, (char *)answer, anslen);
        n = res_send(buf, n, answer, anslen);
        if (n < 0) {
#ifdef DEBUG
                if (_res.options & RES_DEBUG)
                        printf("res_query: send error\n");
#endif
                h_errno = TRY_AGAIN;
                return (n);
        }

        hp = (HEADER *) answer;
        if (hp->rcode != NOERROR || ntohs(hp->ancount) == 0) {
#ifdef DEBUG
                if (_res.options & RES_DEBUG)
                        printf("rcode = %d, ancount=%d\n", hp->rcode,
                            ntohs(hp->ancount));
#endif
                switch (hp->rcode) {
                        case NXDOMAIN:
                                h_errno = HOST_NOT_FOUND;
                                break;
                        case SERVFAIL:
                                h_errno = TRY_AGAIN;
                                break;
                        case NOERROR:
```

```
125                                 h_errno = NO_DATA;
126                                 break;
127                         case FORMERR:
128                         case NOTIMP:
129                         case REFUSED:
130                         default:
131                                 h_errno = NO_RECOVERY;
132                                 break;
133                 }
134                 return (-1);
135         }
136         if (hp->rcode == NOERROR && ntohs(hp->ancount) > 0)
137                 h_errno = 0;
138         return (n);
139 }
```

unchanged_portion_omitted_

```
269 char *
270 hostalias(name)
271         register char *name;
272 {
273         register char *C1, *C2;
274         FILE *fp;
275         char *file;
274         char *file, *getenv(), *strcpy(), *strncpy();
276         char buf[BUFSIZ];
277         static char abuf[MAXDNAME];

279         file = getenv("HOSTALIASES");
280         if (file == NULL || (fp = fopen(file, "r")) == NULL)
281                 return (NULL);
282         buf[sizeof (buf) - 1] = '\0';
283         while (fgets(buf, sizeof (buf), fp)) {
284                 for (C1 = buf; *C1 && !isspace(*C1); ++C1);
285                 if (!*C1)
286                         break;
287                 *C1 = '\0';
288                 if (!strcasecmp(buf, name)) {
289                         while (isspace(*++C1));
290                         if (!*C1)
291                                 break;
292                         for (C2 = C1 + 1; *C2 && !isspace(*C2); ++C2);
293                         abuf[sizeof (abuf) - 1] = *C2 = '\0';
294                         (void) strncpy(abuf, C1, sizeof (abuf) - 1);
295                         fclose(fp);
296                         return (abuf);
297                 }
298         }
299         fclose(fp);
300         return (NULL);
301 }
```

unchanged_portion_omitted_

```
**********************************************************
   12291 Thu Jul  9 15:59:31 2015
new/usr/src/lib/libresolv/res_send.c
1926 libresolv evades compiler warnings
**********************************************************
```

```
/*
 * 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 2015 Gary Mills
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

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

/*
 * Send query to name server and wait for reply.
 */

#include <sys/param.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/uio.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/nameser.h>
#include <arpa/inet.h>
#include <resolv.h>
#include "crossl.h"
```



```
/*
 * Undocumented external function in libsocket
 */
extern int
_socket(int, int, int);

static int s = -1;      /* socket used for communications */
#if     BSD >= 43
static struct sockaddr no_addr;
#endif /* BSD */


#ifndef FD_SET
#define NFDBITS         32
#define FD_SETSIZE      32
#define FD_SET(n, p)    ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) % NFDBITS)))
#define FD_CLR(n, p)    ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) % NFDBITS)))
#define FD_ISSET(n, p)  ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) % NFDBITS)))
#ifdef SYSV
#define FD_ZERO(p)      (void) memset((void *)(p), 0, sizeof (*(p)))
#define FD_ZERO(p)      memset((void *)(p), 0, sizeof (*(p)))
#else
#define FD_ZERO(p)      bzero((char *)(p), sizeof (*(p)))
#endif
#endif

/*
 * 1247019: Kludge to time out quickly if there is no /etc/resolv.conf
 * and a TCP connection to the local DNS server fails.
 */

static int _confcheck()
{
        int ns;
        struct stat rc_stat;
        struct sockaddr_in ns_sin;


        /* First, we check to see if /etc/resolv.conf exists.
         * If it doesn't, then localhost is mostlikely to be
         * the nameserver.
         */
        if (stat(_PATH_RESCONF, &rc_stat) == -1 && errno == ENOENT) {

                /* Next, we check to see if _res.nsaddr is set to loopback.
                 * If it isn't, it has been altered by the application
                 * explicitly and we then want to bail with success.
                 */
                if (_res.nsaddr.sin_addr.S_un.S_addr == htonl(INADDR_LOOPBACK))

                        /* Lastly, we try to connect to the TCP port of the
                         * nameserver.  If this fails, then we know that
                         * DNS is misconfigured and we can quickly exit.
                         */
                        ns = socket(AF_INET, SOCK_STREAM, 0);
                        IN_SET_LOOPBACK_ADDR(&ns_sin);
                        ns_sin.sin_port = htons(NAMESERVER_PORT);
                        if (connect(ns, (struct sockaddr *) &ns_sin,
                                    sizeof ns_sin) == -1) {
                                close(ns);
                                return(-1);
                        }
                        else {
                                close(ns);
                                return(0);
                        }
```

```
125                 }
126
127                 return(0);
128         }
129
130         return (0);
131 }

133 int
134 res_send(buf, buflen, answer, anslen)
135         char *buf;
136         int buflen;
137         char *answer;
138         int anslen;
139 {
140         register int n;
141         int try, v_circuit, resplen, ns;
142         int gotsomewhere = 0;
143 #if     BSD >= 43
144         int connected = 0;
145 #endif /* BSD */
132         int gotsomewhere = 0, connected = 0;
146         int connreset = 0;
147         u_short id, len;
148         char *cp;
149         fd_set dsmask;
150         struct timeval timeout;
151         HEADER *hp = (HEADER *) buf;
152         HEADER *anhp = (HEADER *) answer;
153         struct iovec iov[2];
154         int terrno = ETIMEDOUT;
155         char junk[512];

157 #ifdef DEBUG
158         if (_res.options & RES_DEBUG) {
159                 printf("res_send()\n");
160                 p_query(buf);
161         }
162 #endif
163         if (!(_res.options & RES_INIT))
164                 if (res_init() == -1) {
165                         return (-1);
166                 }

168         /* 1247019: Check to see if we can bailout quickly. */
169         if (_confcheck() == -1)
170             return(-1);

172         v_circuit = (_res.options & RES_USEVC) || buflen > PACKETSZ;
173         id = hp->id;
174         /*
175          * Send request, RETRY times, or until successful
176          */
177         for (try = 0; try < _res.retry; try++) {
178                 for (ns = 0; ns < _res.nscount; ns++) {
179 #ifdef DEBUG
180                         if (_res.options & RES_DEBUG)
181                                 printf("Querying server (# %d) address = %s\n",
182                                 ns+1, inet_ntoa(_res.nsaddr_list[ns].sin_addr));
183 #endif
184                 usevc:
185                         if (v_circuit) {
186                                 int truncated = 0;

188                                 /*
189                                  * Use virtual circuit;
```



```
190                                  * at most one attempt per server.
191                                  */
192                                 try = _res.retry;
193                                 if (s < 0) {
194                                         s = _socket(AF_INET, SOCK_STREAM, 0);
195                                         if (s < 0) {
196                                                 terrno = errno;
197 #ifdef DEBUG
198                                                 if (_res.options & RES_DEBUG) {
199                                                 perror("socket (vc) failed");
200                                                 }
201 #endif
202                                                 continue;
203                                         }
204                                         if (connect(s, (struct sockaddr *) &_res
205                                                 sizeof (struct sockaddr)) < 0) {
206                                                 terrno = errno;
207 #ifdef DEBUG
208                                                 if (_res.options & RES_DEBUG) {
209                                                 perror("connect failed");
210                                                 }
211 #endif
212                                                 (void) close(s);
213                                                 s = -1;
214                                                 continue;
215                                         }
216                                 }
217                                 /*
218                                  * Send length & message
219                                  */
220                                 len = htons((u_short)buflen);
221                                 iov[0].iov_base = (caddr_t)&len;
222                                 iov[0].iov_len = sizeof (len);
223                                 iov[1].iov_base = buf;
224                                 iov[1].iov_len = buflen;
225                                 if (writev(s, iov, 2) != sizeof (len) +
226                                                                 buflen) {
227                                         terrno = errno;
228 #ifdef DEBUG
229                                         if (_res.options & RES_DEBUG)
230                                                 perror("write failed");
231 #endif
232                                         (void) close(s);
233                                         s = -1;
234                                         continue;
235                                 }
236                                 /*
237                                  * Receive length & response
238                                  */
239                                 cp = answer;
240                                 len = sizeof (short);
241                                 while (len != 0 && (n = read
242                                         (s, (char *)cp, (int)len)) > 0) {
243                                         cp += n;
244                                         len -= n;
245                                 }
246                                 if (n <= 0) {
247                                         terrno = errno;
248 #ifdef DEBUG
249                                         if (_res.options & RES_DEBUG)
250                                                 perror("read failed");
251 #endif
252                                         (void) close(s);
253                                         s = -1;
254                                 /*
255                                  * A long running process might get its TCP
```

```
256                              * connection reset if the remote server was
257                              * restarted.  Requery the server instead of
258                              * trying a new one.  When there is only one
259                              * server, this means that a query might work
260                              * instead of failing.  We only allow one reset
261                              * per query to prevent looping.
262                              */
263                                     if (terrno == ECONNRESET &&
264                                                     !connreset) {
265                                             connreset = 1;
266                                             ns--;
267                                     }
268                                     continue;
269                             }
270                             cp = answer;
271                             if ((resplen = ntohs(*(u_short *)cp)) >
272                                                             anslen) {
273 #ifdef DEBUG
274                                     if (_res.options & RES_DEBUG)
275                                             fprintf(stderr,
276                                                     "response truncated\n");
277 #endif
278                                     len = anslen;
279                                     truncated = 1;
280                             } else
281                                     len = resplen;
282                             while (len != 0 &&
283                                     (n = read(s, (char *)cp,
284                                                     (int)len)) > 0) {
285                                     cp += n;
286                                     len -= n;
287                             }
288                             if (n <= 0) {
289                                     terrno = errno;
290 #ifdef DEBUG
291                                     if (_res.options & RES_DEBUG)
292                                             perror("read failed");
293 #endif
294                                     (void) close(s);
295                                     s = -1;
296                                     continue;
297                             }
298                             if (truncated) {
299                                     /*
300                                      * Flush rest of answer
301                                      * so connection stays in synch.
302                                      */
303                                     anhp->tc = 1;
304                                     len = resplen - anslen;
305                                     /*
306                                      * set the value of resplen to anslen,
307                                      * this is done because the caller
308                                      * assumes resplen contains the size of
309                                      * message read into the "answer" buffer
310                                      * passed in.
311                                      */
312                                     resplen = anslen;

314                                     while (len != 0) {
315                                             n = (len > sizeof (junk) ?
316                                                     sizeof (junk) : len);
317                                             if ((n = read(s, junk, n)) > 0)
318                                                     len -= n;
319                                             else
320                                                     break;
321                                     }
```



```
322                                     }
323                             } else {
324                                     /*
325                                      * Use datagrams.
326                                      */
327                                     if (s < 0) {
328                                             s = _socket(AF_INET, SOCK_DGRAM, 0);
329                                             if (s < 0) {
330                                                     terrno = errno;
331 #ifdef DEBUG
332                                                     if (_res.options & RES_DEBUG) {
333                                                     perror("socket (dg) failed");
334                                                     }
335 #endif
336                                                     continue;
337                                             }
338                                     }
339 #if     BSD >= 43
340                             /*
341                              * I'm tired of answering this question, so:
342                              * On a 4.3BSD+ machine (client and server,
343                              * actually), sending to a nameserver datagram
344                              * port with no nameserver will cause an
345                              * ICMP port unreachable message to be returned.
346                              * If our datagram socket is "connected" to the
347                              * server, we get an ECONNREFUSED error on the next
348                              * socket operation, and select returns if the
349                              * error message is received.  We can thus detect
350                              * the absence of a nameserver without timing out.
351                              * If we have sent queries to at least two servers,
352                              * however, we don't want to remain connected,
353                              * as we wish to receive answers from the first
354                              * server to respond.
355                              */
356                                     if (_res.nscount == 1 ||
357                                                     (try == 0 && ns == 0)) {
358                                             /*
359                                              * Don't use connect if we might
360                                              * still receive a response
361                                              * from another server.
362                                              */
363                                             if (connected == 0) {
364                                                     if (connect(s,
365                                                     (struct sockaddr *) &_res.nsaddr
366                                                     sizeof (struct sockaddr)) < 0) {
367 #ifdef DEBUG
368                                                             if (_res.options &
369                                                                     RES_DEBUG) {
370                                                             perror("connect");
371                                                             }
372 #endif
373                                                             continue;
374                                                     }
375                                                     connected = 1;
376                                             }
377                                             if (send(s, buf, buflen, 0) != buflen) {
378 #ifdef DEBUG
379                                                     if (_res.options & RES_DEBUG)
380                                                             perror("send");
381 #endif
382                                                     continue;
383                                             }
384                                     } else {
385                                             /*
386                                              * Disconnect if we want to listen for
387                                              * responses from more than one server.
```

```
388                                          */
389                                         if (connected) {
390                                                 (void) connect(s, &no_addr,
391                                                         sizeof (no_addr));
392                                                 connected = 0;
393                                         }
394 #endif /* BSD */
395                                         if (sendto(s, buf, buflen, 0,
396                                                 (struct sockaddr *) &_res.nsaddr
397                                         sizeof (struct sockaddr)) != buflen) {
398 #ifdef DEBUG
399                                                 if (_res.options & RES_DEBUG)
400                                                         perror("sendto");
401 #endif
402                                                 continue;
403                                         }
404 #if     BSD >= 43
405                                 }
406 #endif

408                                 /*
409                                  * Wait for reply
410                                  */
411                                 timeout.tv_sec = (_res.retrans << try);
412                                 if (try > 0)
413                                         timeout.tv_sec /= _res.nscount;
414                                 if (timeout.tv_sec <= 0)
415                                         timeout.tv_sec = 1;
416                                 timeout.tv_usec = 0;
417 wait:
418                                 FD_ZERO(&dsmask);
419                                 FD_SET(s, &dsmask);
420                                 n = select(s+1, &dsmask, (fd_set *)NULL,
421                                         (fd_set *)NULL, &timeout);
422                                 if (n < 0) {
423 #ifdef DEBUG
424                                         if (_res.options & RES_DEBUG)
425                                                 perror("select");
426 #endif
427                                         continue;
428                                 }
429                                 if (n == 0) {
430                                         /*
431                                          * timeout
432                                          */
433 #ifdef DEBUG
434                                         if (_res.options & RES_DEBUG)
435                                                 printf("timeout\n");
436 #endif
437 #if BSD >= 43
438                                         gotsomewhere = 1;
439 #endif
440                                         continue;
441                                 }
442                                 if ((resplen = recv(s, answer, anslen, 0))
443                                                                         <= 0) {
444 #ifdef DEBUG
445                                         if (_res.options & RES_DEBUG)
446                                                 perror("recvfrom");
447 #endif
448                                         continue;
449                                 }
450                                 gotsomewhere = 1;
451                                 if (id != anhp->id) {
452                                         /*
453                                          * response from old query, ignore it
```



```
454                                          */
455 #ifdef DEBUG
456                                         if (_res.options & RES_DEBUG) {
457                                                 printf("old answer:\n");
458                                                 p_query(answer);
459                                         }
460 #endif
461                                         goto wait;
462                                 }
463                                 if (!(_res.options & RES_IGNTC) && anhp->tc) {
464                                         /*
465                                          * get rest of answer;
466                                          * use TCP with same server.
467                                          */
468 #ifdef DEBUG
469                                         if (_res.options & RES_DEBUG)
470                                                 printf("truncated answer\n");
471 #endif
472                                         (void) close(s);
473                                         s = -1;
474                                         v_circuit = 1;
475                                         goto usevc;
476                                 }
477                         }
478 #ifdef DEBUG
479                         if (_res.options & RES_DEBUG) {
480                                 printf("got answer:\n");
481                                 p_query(answer);
482                         }
483 #endif
484                 /*
485                  * If using virtual circuits, we assume that the first server
486                  * is preferred * over the rest (i.e. it is on the local
487                  * machine) and only keep that one open.
488                  * If we have temporarily opened a virtual circuit,
489                  * or if we haven't been asked to keep a socket open,
490                  * close the socket.
491                  */
492                         if ((v_circuit &&
493                                 ((_res.options & RES_USEVC) == 0 || ns != 0)) ||
494                                 (_res.options & RES_STAYOPEN) == 0) {
495                                 (void) close(s);
496                                 s = -1;
497                         }
498                         return (resplen);
499                 }
500         }
501         if (s >= 0) {
502                 (void) close(s);
503                 s = -1;
504         }
505         if (v_circuit == 0)
506                 if (gotsomewhere == 0)
507                         errno = ECONNREFUSED;   /* no nameservers found */
508                 else
509                         errno = ETIMEDOUT;      /* no answer obtained */
510         else
511                 errno = terrno;
512         return (-1);
513 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    1727 Thu Jul  9 15:59:32 2015
new/usr/src/lib/libresolv/res_sethost.c
1926 libresolv evades compiler warnings
**********************************************************
/*
 * 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 2015 Gary Mills
 * Copyright 2008 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

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

/*
 * University Copyright- Copyright (c) 1982, 1986, 1988
 * The Regents of the University of California
 * All Rights Reserved
 *
 * University Acknowledgment- Portions of this document are derived from
 * software developed by the University of California, Berkeley, and its
 * contributors.
 */

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

#include <sys/types.h>
#include <arpa/nameser.h>
#include <netinet/in.h>
#include <resolv.h>
#include "crossl.h"

void
res_sethostent(stayopen)
int stayopen;
{
        if (stayopen)
                _res.options |= RES_STAYOPEN | RES_USEVC;
}
_____unchanged_portion_omitted_
```