```
**********************************************************
   34869 Wed Apr  9 14:17:36 2014
new/usr/src/lib/libnsl/rpc/rpcb_clnt.c
4729 __rpcb_findaddr_timed should try rpcbind protocol 4 first
**********************************************************
```

```
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License, Version 1.0 only
   6  * (the "License").  You may not use this file except in compliance
   7  * with the License.
   8  *
   9  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
  10  * or http://www.opensolaris.org/os/licensing.
  11  * See the License for the specific language governing permissions
  12  * and limitations under the License.
  13  *
  14  * When distributing Covered Code, include this CDDL HEADER in each
  15  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  16  * If applicable, add the following below this CDDL HEADER, with the
  17  * fields enclosed by brackets "[]" replaced with your own identifying
  18  * information: Portions Copyright [yyyy] [name of copyright owner]
  19  *
  20  * CDDL HEADER END
  21  */

  23 /*
  24  * Copyright (c) 2014 Gary Mills
  25  * Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
  26  * Use is subject to license terms.
  27  */

  29 /* Copyright (c) 1983, 1984, 1985, 1986, 1987, 1988, 1989 AT&T */
  30 /* All Rights Reserved */
  31 /*
  32  * Portions of this source code were derived from Berkeley
  33  * 4.3 BSD under license from the Regents of the University of
  34  * California.
  35  */

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

  37 /*
  38  * interface to rpcbind rpc service.
  39  */

  41 #include "mt.h"
  42 #include "rpc_mt.h"
  43 #include <assert.h>
  44 #include <rpc/rpc.h>
  45 #include <rpc/rpcb_prot.h>
  46 #include <netconfig.h>
  47 #include <netdir.h>
  48 #include <rpc/nettype.h>
  49 #include <syslog.h>
  50 #ifdef PORTMAP
  51 #include <netinet/in.h>         /* FOR IPPROTO_TCP/UDP definitions */
  52 #include <rpc/pmap_prot.h>
  53 #endif
  54 #ifdef ND_DEBUG
  55 #include <stdio.h>
  56 #endif
  57 #include <sys/utsname.h>
  58 #include <errno.h>
  59 #include <stdlib.h>
```



```
  60 #include <string.h>
  61 #include <unistd.h>

  63 static struct timeval tottimeout = { 60, 0 };
  64 static const struct timeval rmttimeout = { 3, 0 };
  65 static struct timeval rpcbrmttime = { 15, 0 };

  67 extern bool_t xdr_wrapstring(XDR *, char **);

  69 static const char nullstring[] = "\000";

  71 extern CLIENT *_clnt_tli_create_timed(int, const struct netconfig *,
  72                         struct netbuf *, rpcprog_t, rpcvers_t, uint_t, uint_t,
  73                         const struct timeval *);

  75 static CLIENT *_getclnthandle_timed(char *, struct netconfig *, char **,
  76                         struct timeval *);


  79 /*
  80  * The life time of a cached entry should not exceed 5 minutes
  81  * since automountd attempts an unmount every 5 minutes.
  82  * It is arbitrarily set a little lower (3 min = 180 sec)
  83  * to reduce the time during which an entry is stale.
  84  */
  85 #define CACHE_TTL 180
  86 #define CACHESIZE 6

  88 struct address_cache {
  89         char *ac_host;
  90         char *ac_netid;
  91         char *ac_uaddr;
  92         struct netbuf *ac_taddr;
  93         struct address_cache *ac_next;
  94         time_t ac_maxtime;
  95 };
_____unchanged_portion_omitted_

 439 /*
 440  * This routine will return a client handle that is connected to the local
 441  * rpcbind. Returns NULL on error and free's everything.
 442  */
 443 static CLIENT *
 444 local_rpcb(void)
 445 {
 446         static struct netconfig *loopnconf;
 447         static char *hostname;
 448         extern mutex_t loopnconf_lock;

 450 /* VARIABLES PROTECTED BY loopnconf_lock: loopnconf */
 451         (void) mutex_lock(&loopnconf_lock);
 452         if (loopnconf == NULL) {
 453                 struct utsname utsname;
 454                 struct netconfig *nconf, *tmpnconf = NULL;
 455                 void *nc_handle;

 457                 if (hostname == NULL) {
 458 #if defined(__i386) && !defined(__amd64)
 459                         if ((_nuname(&utsname) == -1) ||
 460                             ((hostname = strdup(utsname.nodename)) == NULL)) {
 461 #else
 462                         if ((uname(&utsname) == -1) ||
 463                             ((hostname = strdup(utsname.nodename)) == NULL)) {
 464 #endif
 464                             ((hostname = strdup(utsname.nodename)) == NULL)) {
 465                                 syslog(LOG_ERR, "local_rpcb : strdup failed.");
```

```
466                                 rpc_createerr.cf_stat = RPC_UNKNOWNHOST;
467                                 (void) mutex_unlock(&loopnconf_lock);
468                                 return (NULL);
469                         }
470                 }
471                 nc_handle = setnetconfig();
472                 if (nc_handle == NULL) {
473                         /* fails to open netconfig file */
474                         rpc_createerr.cf_stat = RPC_UNKNOWNPROTO;
475                         (void) mutex_unlock(&loopnconf_lock);
476                         return (NULL);
477                 }
478                 while (nconf = getnetconfig(nc_handle)) {
479                         if (strcmp(nconf->nc_protofmly, NC_LOOPBACK) == 0) {
480                                 tmpnconf = nconf;
481                                 if (nconf->nc_semantics == NC_TPI_CLTS)
482                                         break;
483                         }
484                 }
485                 if (tmpnconf == NULL) {
486                         rpc_createerr.cf_stat = RPC_UNKNOWNPROTO;
487                         (void) mutex_unlock(&loopnconf_lock);
488                         return (NULL);
489                 }
490                 loopnconf = getnetconfigent(tmpnconf->nc_netid);
491                 /* loopnconf is never freed */
492                 (void) endnetconfig(nc_handle);
493         }
494         (void) mutex_unlock(&loopnconf_lock);
495         return (getclnthandle(hostname, loopnconf, NULL));
496 }
_____unchanged_portion_omitted_


688 /*
689  * An internal function which optimizes rpcb_getaddr function.  It returns
690  * the universal address of the remote service or NULL.  It also optionally
689  * An internal function which optimizes rpcb_getaddr function.  It also
691  * returns the client handle that it uses to contact the remote rpcbind.
692  *
693  * The algorithm used: First try version 4.  Then try version 3 (svr4).
694  * Finally, if the transport is TCP or UDP, try version 2 (portmap).
695  * We assume that version 4 is now available on many machines on the network.
692  * The algorithm used: If the transports is TCP or UDP, it first tries
693  * version 2 (portmap), 4 and then 3 (svr4).  This order should be
694  * changed in the next OS release to 4, 2 and 3.  We are assuming that by
695  * that time, version 4 would be available on many machines on the network.
696  * With this algorithm, we get performance as well as a plan for
697  * obsoleting version 2.
698  *
699  * For all other transports, the algorithm remains as 4 and then 3.
700  *
699  * XXX: Due to some problems with t_connect(), we do not reuse the same client
700  * handle for COTS cases and hence in these cases we do not return the
701  * client handle.  This code will change if t_connect() ever
702  * starts working properly.  Also look under clnt_vc.c.
703  */
704 struct netbuf *
705 __rpcb_findaddr_timed(rpcprog_t program, rpcvers_t version,
706         struct netconfig *nconf, char *host, CLIENT **clpp, struct timeval *tp)
707 {
708         static bool_t check_rpcbind = TRUE;
709         CLIENT *client = NULL;
710         RPCB parms;
711         enum clnt_stat clnt_st;
712         char *ua = NULL;
```



```
713         uint_t vers;
714         struct netbuf *address = NULL;
715         void *handle;
716         rpcb_entry_list_ptr relp = NULL;
717         bool_t tmp_client = FALSE;
717         uint_t start_vers = RPCBVERS4;

719         /* parameter checking */
720         if (nconf == NULL) {
721                 rpc_createerr.cf_stat = RPC_UNKNOWNPROTO;
722                 return (NULL);
723         }

725         parms.r_addr = NULL;

727         /*
728          * Use default total timeout if no timeout is specified.
729          */
730         if (tp == NULL)
731                 tp = &tottimeout;

733 #ifdef PORTMAP
734         /* Try version 2 for TCP or UDP */
735         if (strcmp(nconf->nc_protofmly, NC_INET) == 0) {
736                 ushort_t port = 0;
737                 struct netbuf remote;
738                 uint_t pmapvers = 2;
739                 struct pmap pmapparms;

733         /*
742                  * Try UDP only - there are some portmappers out
743                  * there that use UDP only.
744                  */
745                 if (strcmp(nconf->nc_proto, NC_TCP) == 0) {
746                         struct netconfig *newnconf;
747                         void *handle;

749                         if ((handle = __rpc_setconf("udp")) == NULL) {
750                                 rpc_createerr.cf_stat = RPC_UNKNOWNPROTO;
751                                 return (NULL);
752                         }

754                         /*
755                          * The following to reinforce that you can
756                          * only request for remote address through
757                          * the same transport you are requesting.
758                          * ie. requesting unversial address
759                          * of IPv4 has to be carried through IPv4.
760                          * Can't use IPv6 to send out the request.
761                          * The mergeaddr in rpcbind can't handle
762                          * this.
763                          */
764                         for (;;) {
765                                 if ((newnconf = __rpc_getconf(handle))
766                                                                 == NULL) {
767                                         __rpc_endconf(handle);
768                                         rpc_createerr.cf_stat =
769                                             RPC_UNKNOWNPROTO;
770                                         return (NULL);
771                                 }
772                                 /*
773                                  * here check the protocol family to
774                                  * be consistent with the request one
775                                  */
776                                 if (strcmp(newnconf->nc_protofmly,
777                                     nconf->nc_protofmly) == NULL)
```

```
778                                     break;
779                             }

781                             client = _getclnthandle_timed(host, newnconf,
782                                     &parms.r_addr, tp);
783                             __rpc_endconf(handle);
784                     } else {
785                             client = _getclnthandle_timed(host, nconf,
786                                     &parms.r_addr, tp);
787                     }
788                     if (client == NULL)
789                             return (NULL);

791                     /*
792                      * Set version and retry timeout.
793                      */
794                     CLNT_CONTROL(client, CLSET_RETRY_TIMEOUT, (char *)&rpcbrmttime);
795                     CLNT_CONTROL(client, CLSET_VERS, (char *)&pmapvers);

797                     pmapparms.pm_prog = program;
798                     pmapparms.pm_vers = version;
799                     pmapparms.pm_prot = strcmp(nconf->nc_proto, NC_TCP) ?
800                                     IPPROTO_UDP : IPPROTO_TCP;
801                     pmapparms.pm_port = 0;  /* not needed */
802                     clnt_st = CLNT_CALL(client, PMAPPROC_GETPORT,
803                                     (xdrproc_t)xdr_pmap, (caddr_t)&pmapparms,
804                                     (xdrproc_t)xdr_u_short, (caddr_t)&port,
805                                     *tp);
806                     if (clnt_st != RPC_SUCCESS) {
807                             if ((clnt_st == RPC_PROGVERSMISMATCH) ||
808                                 (clnt_st == RPC_PROGUNAVAIL))
809                                     goto try_rpcbind; /* Try different versions */
810                             rpc_createerr.cf_stat = RPC_PMAPFAILURE;
811                             clnt_geterr(client, &rpc_createerr.cf_error);
812                             goto error;
813                     } else if (port == 0) {
814                             address = NULL;
815                             rpc_createerr.cf_stat = RPC_PROGNOTREGISTERED;
816                             goto error;
817                     }
818                     port = htons(port);
819                     CLNT_CONTROL(client, CLGET_SVC_ADDR, (char *)&remote);
820                     if (((address = malloc(sizeof (struct netbuf))) == NULL) ||
821                         ((address->buf = malloc(remote.len)) == NULL)) {
822                             rpc_createerr.cf_stat = RPC_SYSTEMERROR;
823                             clnt_geterr(client, &rpc_createerr.cf_error);
824                             if (address) {
825                                     free(address);
826                                     address = NULL;
827                             }
828                             goto error;
829                     }
830                     (void) memcpy(address->buf, remote.buf, remote.len);
831                     (void) memcpy(&address->buf[sizeof (short)], &port,
832                                                             sizeof (short));
833                     address->len = address->maxlen = remote.len;
834                     goto done;
835             }
836     #endif

838     try_rpcbind:
839             /*
734              * Check if rpcbind is up.  This prevents needless delays when
735              * accessing applications such as the keyserver while booting
736              * disklessly.
737              */
```



```
738             if (check_rpcbind && strcmp(nconf->nc_protofmly, NC_LOOPBACK) == 0) {
739                     if (!__rpcbind_is_up()) {
740                             rpc_createerr.cf_stat = RPC_PMAPFAILURE;
741                             rpc_createerr.cf_error.re_errno = 0;
742                             rpc_createerr.cf_error.re_terrno = 0;
743                             goto error;
744                     }
745                     check_rpcbind = FALSE;
746             }

748             /*
749                      * First try version 4.
855                      * Now we try version 4 and then 3.
856                      * We also send the remote system the address we used to
857                      * contact it in case it can help to connect back with us
750                      */
751             parms.r_prog = program;
752             parms.r_vers = version;
753             parms.r_owner = (char *)&nullstring[0]; /* not needed; */
754             /* just for xdring */
755             parms.r_netid = nconf->nc_netid; /* not really needed */

757             /*
758              * If a COTS transport is being used, try getting address via CLTS
759              * transport.  This works only with version 4.
760              */
761             if (nconf->nc_semantics == NC_TPI_COTS_ORD ||
762                 nconf->nc_semantics == NC_TPI_COTS) {
763                     handle = __rpc_setconf("datagram_v");
764             } else {
765                     handle = __rpc_setconf(nconf->nc_proto);
766             }

768             if (handle != NULL) {
871                     void *handle;
769                     struct netconfig *nconf_clts;
873                     rpcb_entry_list_ptr relp = NULL;

875                     if (client == NULL) {
876                             /* This did not go through the above PORTMAP/TCP code */
877                             if ((handle = __rpc_setconf("datagram_v")) != NULL) {
771                     while ((nconf_clts = __rpc_getconf(handle))
772                         != NULL) {
773                             if (strcmp(nconf_clts->nc_protofmly,
774                                 nconf->nc_protofmly) != 0) {
775                                     continue;
776                             }
777                             client = _getclnthandle_timed(host,
778                                 nconf_clts, &parms.r_addr,
779                                 tp);
780                             break;
781                     }
782                     __rpc_endconf(handle);
783             }
784             if (client != NULL) {

786                     if (nconf->nc_semantics == NC_TPI_COTS_ORD ||
787                         nconf->nc_semantics == NC_TPI_COTS)
788                             tmp_client = TRUE;

790                     /* Set rpcbind version 4 */
891                             if (client == NULL)
892                                     goto regular_rpcbind;   /* Go the regular way */
893                     } else {
894                             /* This is a UDP PORTMAP handle.  Change to version 4 */
791                     vers = RPCBVERS4;
```

```
792                 CLNT_CONTROL(client, CLSET_VERS, (char *)&vers);
897                 }
794                 /*
795                  * We also send the remote system the address we used to
796                  * contact it in case it can help it connect back with us
797                  */
798                 if (parms.r_addr == NULL) {
799                         parms.r_addr = strdup(""); /* for XDRing */
800                         if (parms.r_addr == NULL) {
801                                 syslog(LOG_ERR, "__rpcb_findaddr_timed: "
802                                     "strdup failed.");
803                                 rpc_createerr.cf_stat = RPC_SYSTEMERROR;
804                                 address = NULL;
805                                 goto error;
806                         }
807                 }
809                 CLNT_CONTROL(client, CLSET_RETRY_TIMEOUT,
810                     (char *)&rpcbrmttime);
913                 CLNT_CONTROL(client, CLSET_RETRY_TIMEOUT, (char *)&rpcbrmttime);
812                 clnt_st = CLNT_CALL(client, RPCBPROC_GETADDRLIST,
813                     (xdrproc_t)xdr_rpcb, (char *)&parms,
814                     (xdrproc_t)xdr_rpcb_entry_list_ptr,
815                     (char *)&relp, *tp);
816                 switch (clnt_st) {
817                 case RPC_SUCCESS:
919                 if (clnt_st == RPC_SUCCESS) {
818                         if (address = got_entry(relp, nconf)) {
819                                 xdr_free((xdrproc_t)xdr_rpcb_entry_list_ptr,
820                                     (char *)&relp);
821                                 goto done;
822                         }
823                         /* Entry not found for this transport */
824                         xdr_free((xdrproc_t)xdr_rpcb_entry_list_ptr,
825                             (char *)&relp);
826                         /*
827                          * XXX: should have perhaps returned with error but
828                          * since the remote machine might not always be able
829                          * to send the address on all transports, we try the
830                          * regular way with version 3, then 2
932                          * regular way with regular_rpcbind
831                          */
832                         /* Try the next version */
833                         break;
834                 case RPC_PROGVERSMISMATCH:
835                 case RPC_PROGUNAVAIL:
836                         /* Try the next version */
837                         break;
838                 default:
934                         goto regular_rpcbind;
935                 } else if ((clnt_st == RPC_PROGVERSMISMATCH) ||
936                             (clnt_st == RPC_PROGUNAVAIL)) {
937                         start_vers = RPCBVERS;  /* Try version 3 now */
938                         goto regular_rpcbind; /* Try different versions */
939                 } else {
839                         rpc_createerr.cf_stat = RPC_PMAPFAILURE;
840                         clnt_geterr(client, &rpc_createerr.cf_error);
841                         goto error;
842                         break;
843                 }
844         } /* End of version 4 */
944         }
846         /*
```



```
847          * Try version 3
848          */
946 regular_rpcbind:
850         /* Now the same transport is to be used to get the address */
851         if (client && ((nconf->nc_semantics == NC_TPI_COTS_ORD) ||
852             (nconf->nc_semantics == NC_TPI_COTS))) {
853                 /* A CLTS type of client - destroy it */
854                 CLNT_DESTROY(client);
855                 client = NULL;
856                 free(parms.r_addr);
857                 parms.r_addr = NULL;
858         }
860         if (client == NULL) {
861                 client = _getclnthandle_timed(host, nconf, &parms.r_addr, tp);
960                 if (client == NULL) {
961                         address = NULL;
962                         goto error;
862         }
863         if (client != NULL) {
864                 tmp_client = FALSE;
964         }
865                 if (parms.r_addr == NULL) {
866                         parms.r_addr = strdup("");      /* for XDRing */
867                         if (parms.r_addr == NULL) {
868                                 syslog(LOG_ERR, "__rpcb_findaddr_timed: "
869                                     "strdup failed.");
870                                 address = NULL;
871                                 rpc_createerr.cf_stat = RPC_SYSTEMERROR;
872                                 goto error;
873                         }
874                 }
876                 CLNT_CONTROL(client, CLSET_RETRY_TIMEOUT,
877                     (char *)&rpcbrmttime);
878                 vers = RPCBVERS; /* Set the version */
976         /* First try from start_vers and then version 3 (RPCBVERS) */
978         CLNT_CONTROL(client, CLSET_RETRY_TIMEOUT, (char *)&rpcbrmttime);
979         for (vers = start_vers;  vers >= RPCBVERS; vers--) {
980                 /* Set the version */
879                 CLNT_CONTROL(client, CLSET_VERS, (char *)&vers);
880                 clnt_st = CLNT_CALL(client, RPCBPROC_GETADDR,
881                     (xdrproc_t)xdr_rpcb, (char *)&parms,
882                     (xdrproc_t)xdr_wrapstring,
883                     (char *)&ua, *tp);
884                 switch (clnt_st) {
885                 case RPC_SUCCESS:
886                         if ((ua != NULL) && (ua[0] != '\0')) {
887                                 address = uaddr2taddr(nconf, ua);
888 #ifdef ND_DEBUG
889                                 fprintf(stderr, "\tRemote address is [%s]\n",
890                                     ua);
891 #endif
892                                 xdr_free((xdrproc_t)xdr_wrapstring,
893                                     (char *)&ua);
894                         } else if (ua != NULL) {
986                 if (clnt_st == RPC_SUCCESS) {
987                         if ((ua == NULL) || (ua[0] == NULL)) {
988                                 if (ua != NULL)
895                                 xdr_free(xdr_wrapstring, (char *)&ua);
896                         }
898                         if (ua != NULL && address != NULL) {
899                                 goto done;
```

```
                        } else if (address == NULL) {
                                /* We don't know about your universal addr */
                                /* address unknown */
                                rpc_createerr.cf_stat = RPC_PROGNOTREGISTERED;
                                goto error;
                        }
                        address = uaddr2taddr(nconf, ua);
#ifdef ND_DEBUG
                        fprintf(stderr, "\tRemote address is [%s]\n", ua);
                        if (!address)
                                fprintf(stderr,
                                    "\tCouldn't resolve remote address!\n");
#endif
                                rpc_createerr.cf_stat = RPC_PROGNOTREGISTERED;
                        xdr_free((xdrproc_t)xdr_wrapstring, (char *)&ua);

                        if (!address) {
                                /* We don't know about your universal address */
                                rpc_createerr.cf_stat = RPC_N2AXLATEFAILURE;
                                goto error;
                        }
                        /* Try the next version */
                        break;
                case RPC_PROGVERSMISMATCH:
                        clnt_geterr(client, &rpc_createerr.cf_error);
                        if (rpc_createerr.cf_error.re_vers.low > RPCBVERS4)
                        goto done;
                }
                if (clnt_st == RPC_PROGVERSMISMATCH) {
                        struct rpc_err rpcerr;

                        clnt_geterr(client, &rpcerr);
                        if (rpcerr.re_vers.low > RPCBVERS4)
                                goto error;  /* a new version, can't handle */
                        /* Try the next version */
                        break;
                case RPC_PROGUNAVAIL:
                        /* Try the next version */
                        break;
                default:
                        clnt_geterr(client, &rpc_createerr.cf_error);
                        rpc_createerr.cf_stat = RPC_PMAPFAILURE;
                } else if (clnt_st != RPC_PROGUNAVAIL) {
                        /* Cant handle this error */
                        goto error;
                        break;
                }
        } else {
                address = NULL;
        } /* End of version 3 */

        /*
         * Try version 2
         */

#ifdef PORTMAP
        /* Try version 2 for TCP or UDP */
        if (strcmp(nconf->nc_protofmly, NC_INET) == 0) {
                ushort_t port = 0;
                struct netbuf remote;
                uint_t pmapvers = 2;
                struct pmap pmapparms;

                /*
                 * Try UDP only - there are some portmappers out
                 * there that use UDP only.
```



```
                 */
                if (strcmp(nconf->nc_proto, NC_TCP) == 0) {
                        struct netconfig *newnconf;

                        if (client) {
                                CLNT_DESTROY(client);
                                client = NULL;
                                free(parms.r_addr);
                                parms.r_addr = NULL;
                        }
                        if ((handle = __rpc_setconf("udp")) == NULL) {
                                rpc_createerr.cf_stat = RPC_UNKNOWNPROTO;
                                return (NULL);
                        }

                        /*
                         * The following to reinforce that you can
                         * only request for remote address through
                         * the same transport you are requesting.
                         * ie. requesting unversial address
                         * of IPv4 has to be carried through IPv4.
                         * Can't use IPv6 to send out the request.
                         * The mergeaddr in rpcbind can't handle
                         * this.
                         */
                        for (;;) {
                                if ((newnconf = __rpc_getconf(handle))
                                    == NULL) {
                                        __rpc_endconf(handle);
                                        rpc_createerr.cf_stat =
                                            RPC_UNKNOWNPROTO;
                                        return (NULL);
                                }
                                /*
                                 * here check the protocol family to
                                 * be consistent with the request one
                                 */
                                if (strcmp(newnconf->nc_protofmly,
                                    nconf->nc_protofmly) == NULL)
                                        break;
                        }

                        client = _getclnthandle_timed(host, newnconf,
                            &parms.r_addr, tp);
                        __rpc_endconf(handle);
                }
                if (client == NULL)
                        return (NULL);

                if (strcmp(nconf->nc_proto, NC_TCP) == 0)
                        tmp_client = TRUE;

                /*
                 * Set version and retry timeout.
                 */
                CLNT_CONTROL(client, CLSET_RETRY_TIMEOUT, (char *)&rpcbrmttime);
                CLNT_CONTROL(client, CLSET_VERS, (char *)&pmapvers);

                pmapparms.pm_prog = program;
                pmapparms.pm_vers = version;
                pmapparms.pm_prot = strcmp(nconf->nc_proto, NC_TCP) ?
                    IPPROTO_UDP : IPPROTO_TCP;
                pmapparms.pm_port = 0;  /* not needed */
                clnt_st = CLNT_CALL(client, PMAPPROC_GETPORT,
                    (xdrproc_t)xdr_pmap, (caddr_t)&pmapparms,
                    (xdrproc_t)xdr_u_short, (caddr_t)&port,
```

```
1011                        *tp);
1012                if (clnt_st != RPC_SUCCESS) {
1013                        rpc_createerr.cf_stat = RPC_PMAPFAILURE;
1014                        clnt_geterr(client, &rpc_createerr.cf_error);
1015                        goto error;
1016                } else if (port == 0) {
1017                        address = NULL;
1023        if ((address == NULL) || (address->len == 0)) {
1018                        rpc_createerr.cf_stat = RPC_PROGNOTREGISTERED;
1019                        goto error;
1020                }
1021                port = htons(port);
1022                CLNT_CONTROL(client, CLGET_SVC_ADDR, (char *)&remote);
1023                if (((address = malloc(sizeof (struct netbuf))) == NULL) ||
1024                    ((address->buf = malloc(remote.len)) == NULL)) {
1025                        rpc_createerr.cf_stat = RPC_SYSTEMERROR;
1026                        clnt_geterr(client, &rpc_createerr.cf_error);
1027                        if (address) {
1028                                free(address);
1029                                address = NULL;
1030                        }
1031                        goto error;
1032                }
1033                (void) memcpy(address->buf, remote.buf, remote.len);
1034                (void) memcpy(&address->buf[sizeof (short)], &port,
1035                    sizeof (short));
1036                address->len = address->maxlen = remote.len;
1037                goto done;
1038        }
1039 #endif

1041 error:
1042          /* Return NULL address and NULL client */
1043          if (client) {
1044                  CLNT_DESTROY(client);
1045                  client = NULL;
1046          }

1048 done:
1049          /* Return an address and optional client */
1050          if (tmp_client) {
1051                  /* This client is the temporary one */
1034          if (nconf->nc_semantics != NC_TPI_CLTS) {
1035                  /* This client is the connectionless one */
1052                  if (client) {
1053                          CLNT_DESTROY(client);
1054                          client = NULL;
1055                  }
1056          }
1057          if (clpp) {
1058                  *clpp = client;
1059          } else if (client) {
1060                  CLNT_DESTROY(client);
1061          }
1062          if (parms.r_addr)
1063                  free(parms.r_addr);
1064          return (address);
1065 }
```

_____*unchanged_portion_omitted_*