```
**********************************************************
   96672 Fri Apr  1 09:11:12 2016
new/usr/src/lib/libdns_sd/common/dnssd_clientstub.c
6771 end-of-loop code not reached in common/dnssd_clientstub.c
**********************************************************
______unchanged_portion_omitted_

 596 #define deliver_request_bailout(MSG) \
 597     syslog(LOG_WARNING, "dnssd_clientstub deliver_request: %s failed %d (%s)", (
 597     do { syslog(LOG_WARNING, "dnssd_clientstub deliver_request: %s failed %d (%s

 599 static DNSServiceErrorType deliver_request(ipc_msg_hdr *hdr, DNSServiceOp *sdr)
 600 {
 601     uint32_t datalen = hdr->datalen;    // We take a copy here because we're goi
 602     #if defined(USE_TCP_LOOPBACK) || defined(USE_NAMED_ERROR_RETURN_SOCKET)
 603     char *const data = (char *)hdr + sizeof(ipc_msg_hdr);
 604     #endif
 605     dnssd_sock_t listenfd = dnssd_InvalidSocket, errsd = dnssd_InvalidSocket;
 606     DNSServiceErrorType err = kDNSServiceErr_Unknown;   // Default for the "goto
 607     int MakeSeparateReturnSocket = 0;

 609     // Note: need to check hdr->op, not sdr->op.
 610     // hdr->op contains the code for the specific operation we're currently doin
 611     // contains the original parent DNSServiceOp (e.g. for an add_record_request
 612     // add_record_request but the parent sdr->op will be connection_request or r
 613     if (sdr->primary ||
 614         hdr->op == reg_record_request || hdr->op == add_record_request || hdr->o
 615         MakeSeparateReturnSocket = 1;

 617     if (!DNSServiceRefValid(sdr))
 618     {
 619         if (hdr)
 620             free(hdr);
 621         syslog(LOG_WARNING, "dnssd_clientstub deliver_request: invalid DNSServic
 622         return kDNSServiceErr_BadReference;
 623     }

 625     if (!hdr)
 626     {
 627         syslog(LOG_WARNING, "dnssd_clientstub deliver_request: !hdr");
 628         return kDNSServiceErr_Unknown;
 629     }

 631     if (MakeSeparateReturnSocket)
 632     {
 633         #if defined(USE_TCP_LOOPBACK)
 634         {
 635             union { uint16_t s; u_char b[2]; } port;
 636             dnssd_sockaddr_t caddr;
 637             dnssd_socklen_t len = (dnssd_socklen_t) sizeof(caddr);
 638             listenfd = socket(AF_DNSSD, SOCK_STREAM, 0);
 639             if (!dnssd_SocketValid(listenfd)) {
 640                                 deliver_request_bailout("TCP socket");
 641                     }
 639             if (!dnssd_SocketValid(listenfd)) deliver_request_bailout("TCP socke

 643             caddr.sin_family      = AF_INET;
 644             caddr.sin_port        = 0;
 645             caddr.sin_addr.s_addr = inet_addr(MDNS_TCP_SERVERADDR);
 646             if (bind(listenfd, (struct sockaddr*) &caddr, sizeof(caddr)) < 0) {
 647                                 deliver_request_bailout("TCP bind");
 648                     }
 649             if (getsockname(listenfd, (struct sockaddr*) &caddr, &len)   < 0) {
 650                                 deliver_request_bailout("TCP getsockname");
 651                     }
 652                     if (listen(listenfd, 1)
```



```
 653                                 deliver_request_bailout("TCP listen");
 654                     }
 644             if (bind(listenfd, (struct sockaddr*) &caddr, sizeof(caddr)) < 0) de
 645             if (getsockname(listenfd, (struct sockaddr*) &caddr, &len)   < 0) de
 646             if (listen(listenfd, 1)                                              < 0) de
 655             port.s = caddr.sin_port;
 656             data[0] = port.b[0];  // don't switch the byte order, as the
 657             data[1] = port.b[1];  // daemon expects it in network byte order
 658         }
 659         #elif defined(USE_NAMED_ERROR_RETURN_SOCKET)
 660         {
 661             mode_t mask;
 662             int bindresult;
 663             dnssd_sockaddr_t caddr;
 664             listenfd = socket(AF_DNSSD, SOCK_STREAM, 0);
 665             if (!dnssd_SocketValid(listenfd)) {
 666                                 deliver_request_bailout("USE_NAMED_ERROR_RETURN_
 667                     }
 657             if (!dnssd_SocketValid(listenfd)) deliver_request_bailout("USE_NAMED

 669             caddr.sun_family = AF_LOCAL;
 670             // According to Stevens (section 3.2), there is no portable way to
 671             // determine whether sa_len is defined on a particular platform.
 672             #ifndef NOT_HAVE_SA_LEN
 673             caddr.sun_len = sizeof(struct sockaddr_un);
 674             #endif
 675             strcpy(caddr.sun_path, data);
 676             mask = umask(0);
 677             bindresult = bind(listenfd, (struct sockaddr *)&caddr, sizeof(caddr)
 678             umask(mask);
 679             if (bindresult          < 0) {
 680                                 deliver_request_bailout("USE_NAMED_ERROR_RETURN_
 669             if (bindresult          < 0) deliver_request_bailout("USE_NAMED_ERRO
 670             if (listen(listenfd, 1) < 0) deliver_request_bailout("USE_NAMED_ERRO
 681                     }
 682             if (listen(listenfd, 1) < 0) {
 683                                 deliver_request_bailout("USE_NAMED_ERROR_RETURN_
 684                     }
 685         }
 686         #else
 687         {
 688             dnssd_sock_t sp[2];
 689             if (socketpair(AF_DNSSD, SOCK_STREAM, 0, sp) < 0) {
 690                                 deliver_request_bailout("socketpair");
 691                     }
 675             if (socketpair(AF_DNSSD, SOCK_STREAM, 0, sp) < 0) deliver_request_ba
 692             else
 693             {
 694                 errsd    = sp[0];   // We'll read our four-byte error code from
 695                 listenfd = sp[1];   // We'll send sp[1] to the daemon
 696                 #if !defined(__ppc__) && defined(SO_DEFUNCTOK)
 697                 {
 698                     int defunct = 1;
 699                     if (setsockopt(errsd, SOL_SOCKET, SO_DEFUNCTOK, &defunct, si
 700                         syslog(LOG_WARNING, "dnssd_clientstub ConnectToServer: S
 701                 }
 702                 #endif
 703             }
 704         }
 705         #endif
 706     }

 708 #if !defined(USE_TCP_LOOPBACK) && !defined(USE_NAMED_ERROR_RETURN_SOCKET)
 709     // If we're going to make a separate error return socket, and pass it to the
 710     // using sendmsg, then we'll hold back one data byte to go with it.
 711     // On some versions of Unix (including Leopard) sending a control message wi
```

```
712     // any associated data does not work reliably -- e.g. one particular issue w
713     // into is that if the receiving program is in a kqueue loop waiting to be n
714     // of the received message, it doesn't get woken up when the control message
715     if (MakeSeparateReturnSocket || sdr->op == send_bpf)
716         datalen--;      // Okay to use sdr->op when checking for op == send_bpf
717 #endif

719     // At this point, our listening socket is set up and waiting, if necessary,
720     ConvertHeaderBytes(hdr);
721     //syslog(LOG_WARNING, "dnssd_clientstub deliver_request writing %lu bytes",
722     //if (MakeSeparateReturnSocket) syslog(LOG_WARNING, "dnssd_clientstub delive
723 #if TEST_SENDING_ONE_BYTE_AT_A_TIME
724     unsigned int i;
725     for (i=0; i<datalen + sizeof(ipc_msg_hdr); i++)
726     {
727         syslog(LOG_WARNING, "dnssd_clientstub deliver_request writing %d", i);
728         if (write_all(sdr->sockfd, ((char *)hdr)+i, 1) < 0)
729         { syslog(LOG_WARNING, "write_all (byte %u) failed", i); goto cleanup; }
730         usleep(10000);
731     }
732 #else
733     if (write_all(sdr->sockfd, (char *)hdr, datalen + sizeof(ipc_msg_hdr)) < 0)
734     {
735         // write_all already prints an error message if there is an error writin
736         // the socket except for DEFUNCT. Logging here is unnecessary and also w
737         // in the case of DEFUNCT sockets
738         syslog(LOG_INFO, "dnssd_clientstub deliver_request ERROR: write_all(%d,
739                sdr->sockfd, (unsigned long)(datalen + sizeof(ipc_msg_hdr)));
740         goto cleanup;
741     }
742 #endif

744     if (!MakeSeparateReturnSocket)
745         errsd = sdr->sockfd;
746     if (MakeSeparateReturnSocket || sdr->op == send_bpf)    // Okay to use sdr->
747     {
748 #if defined(USE_TCP_LOOPBACK) || defined(USE_NAMED_ERROR_RETURN_SOCKET)
749         // At this point we may wait in accept for a few milliseconds waiting fo
750         // but that's okay -- the daemon should not take more than a few millise
751         // set_waitlimit() ensures we do not block indefinitely just in case som
752         dnssd_sockaddr_t daddr;
753         dnssd_socklen_t len = sizeof(daddr);
754         if ((err = set_waitlimit(listenfd, DNSSD_CLIENT_TIMEOUT)) != kDNSService
755             goto cleanup;
756         errsd = accept(listenfd, (struct sockaddr *)&daddr, &len);
757         if (!dnssd_SocketValid(errsd)) {
758         if (!dnssd_SocketValid(errsd))
759             deliver_request_bailout("accept");
760                 }
761 #else

763         struct iovec vec = { ((char *)hdr) + sizeof(ipc_msg_hdr) + datalen, 1 };
764         struct msghdr msg;
765         struct cmsghdr *cmsg;
766         char cbuf[CMSG_SPACE(4 * sizeof(dnssd_sock_t))];

768         msg.msg_name       = 0;
769         msg.msg_namelen    = 0;
770         msg.msg_iov        = &vec;
771         msg.msg_iovlen     = 1;
772         msg.msg_flags      = 0;
773         if (MakeSeparateReturnSocket || sdr->op == send_bpf)    // Okay to use s
774         {
775             if (sdr->op == send_bpf)
776             {
777                 int i;
```



```
778                 char p[12];     // Room for "/dev/bpf999" with terminating null
779                 for (i=0; i<100; i++)
780                 {
781                     snprintf(p, sizeof(p), "/dev/bpf%d", i);
782                     listenfd = open(p, O_RDWR, 0);
783                     //if (dnssd_SocketValid(listenfd)) syslog(LOG_WARNING, "Send
784                     if (!dnssd_SocketValid(listenfd) && dnssd_errno != EBUSY)
785                         syslog(LOG_WARNING, "Error opening %s %d (%s)", p, dnssd
786                     if (dnssd_SocketValid(listenfd) || dnssd_errno != EBUSY) bre
787                 }
788             }
789             msg.msg_control    = cbuf;
790             msg.msg_controllen = CMSG_LEN(sizeof(dnssd_sock_t));

792             cmsg = CMSG_FIRSTHDR(&msg);
793             cmsg->cmsg_len     = CMSG_LEN(sizeof(dnssd_sock_t));
794             cmsg->cmsg_level   = SOL_SOCKET;
795             cmsg->cmsg_type    = SCM_RIGHTS;
796             *((dnssd_sock_t *)CMSG_DATA(cmsg)) = listenfd;
797         }

799 #if TEST_KQUEUE_CONTROL_MESSAGE_BUG
800         sleep(1);
801 #endif

803 #if DEBUG_64BIT_SCM_RIGHTS
804         syslog(LOG_WARNING, "dnssd_clientstub sendmsg read sd=%d write sd=%d %ld
805                errsd, listenfd, sizeof(dnssd_sock_t), sizeof(void*),
806                sizeof(struct cmsghdr) + sizeof(dnssd_sock_t),
807                CMSG_LEN(sizeof(dnssd_sock_t)), (long)CMSG_SPACE(sizeof(dnssd_soc
808                (long)((char*)CMSG_DATA(cmsg) + 4 - cbuf));
809 #endif // DEBUG_64BIT_SCM_RIGHTS

811         if (sendmsg(sdr->sockfd, &msg, 0) < 0)
812         {
813             syslog(LOG_WARNING, "dnssd_clientstub deliver_request ERROR: sendmsg
814                    errsd, listenfd, dnssd_errno, dnssd_strerror(dnssd_errno));
815             err = kDNSServiceErr_Incompatible;
816             goto cleanup;
817         }

819 #if DEBUG_64BIT_SCM_RIGHTS
820         syslog(LOG_WARNING, "dnssd_clientstub sendmsg read sd=%d write sd=%d oka
821 #endif // DEBUG_64BIT_SCM_RIGHTS

823 #endif
824         // Close our end of the socketpair *before* calling read_all() to get th
825         // Otherwise, if the daemon closes our socket (or crashes), we will have
826         // in read_all() because the socket is not closed (we still have an open
827         // Note: listenfd is overwritten in the case of send_bpf above and that
828         // for send_bpf operation.
829         dnssd_close(listenfd);
830         listenfd = dnssd_InvalidSocket; // Make sure we don't close it a second
831     }

833     // At this point we may wait in read_all for a few milliseconds waiting for
834     // but that's okay -- the daemon should not take more than a few millisecond
835     // set_waitlimit() ensures we do not block indefinitely just in case somethi
836     if (sdr->op == send_bpf)    // Okay to use sdr->op when checking for op == s
837         err = kDNSServiceErr_NoError;
838     else if ((err = set_waitlimit(errsd, DNSSD_CLIENT_TIMEOUT)) == kDNSServiceEr
839     {
840         if (read_all(errsd, (char*)&err, (int)sizeof(err)) < 0)
841             err = kDNSServiceErr_ServiceNotRunning; // On failure read_all will
842         else
843             err = ntohl(err);
```

```
 843     }
 844     //syslog(LOG_WARNING, "dnssd_clientstub deliver_request: retrieved error cod

 846 cleanup:
 847     if (MakeSeparateReturnSocket)
 848     {
 849         if (dnssd_SocketValid(listenfd)) dnssd_close(listenfd);
 850         if (dnssd_SocketValid(errsd)) dnssd_close(errsd);
 851 #if defined(USE_NAMED_ERROR_RETURN_SOCKET)
 852         // syslog(LOG_WARNING, "dnssd_clientstub deliver_request: removing UDS:
 853         if (unlink(data) != 0)
 854             syslog(LOG_WARNING, "dnssd_clientstub WARNING: unlink(\"%s\") failed
 855         // else syslog(LOG_WARNING, "dnssd_clientstub deliver_request: removed U
 856 #endif
 857     }

 859     free(hdr);
 860     return err;
 861 }
```

*unchanged_portion_omitted_*